(9)

Rockwell International

Volume 2

523-1001818-001821
15 March 1977

Rockwell International

appendices

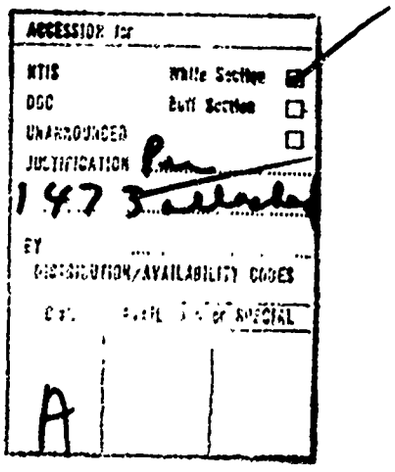
Volume 2

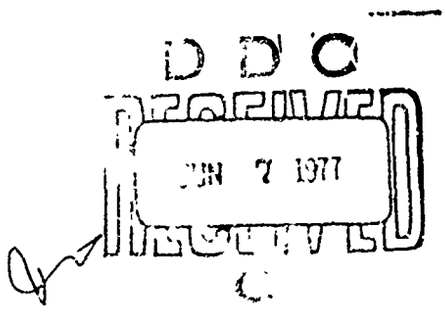
An Architectural Study of Signal Processing Systems and Switched Networks

Submitted to the Defense Communication Agency in partial fullfillment of requirements for contract No. 100-76-C-0070

Collins Government Telecommunications Division
Rockwell International
Newport Beach, California 92663

Printed in the United States of America





UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1 REPORT NUMBER	2 GOVT ACCESSION NO. / 3 RECIPIENT'S CATALOG NUMBER (9)
4 TITLE (and Subtitle) (6) AN ARCHITECTURAL STUDY OF SIGNAL PROCESSING SYSTEMS AND SWITCHED NETWORKS. Final Report, Volume 1 Appendix L; Volume 3 Appendices, Volume 2	5 TYPE OF REPORT & PERIOD COVERED Final Report 16 Aug 76 - 15 Mar 77, / 6 PERFORMING ORG. REPORT NUMBER
7 AUTHOR(s) Volume 1 & 2. Appendices.	8 CONTRACT OR GRANT NUMBER(s) (15) DCA100-76-C-0070
9 PERFORMING ORGANIZATION NAME AND ADDRESS Rockwell International, Collins Radio Group 4311 Jamboree Boulevard Newport Beach, CA 92663	10 PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS PE 33126K Task 15306C
11 CONTROLLING OFFICE NAME AND ADDRESS Defense Communications Engineering Center/R830 (11) 1860 Wiehle Avenue Reston, VA 22090	12 REPORT DATE 15 Mar 77 / 13 NUMBER OF PAGES
14 MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) (12) 199p.	15 SECURITY CLASS. (of this report) UNCLASSIFIED / 15a DECLASSIFICATION/DOWNGRADING SCHEDULE

16 DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17 DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18 SUPPLEMENTARY NOTES

19 KEY WORDS (Continue on reverse side if necessary and identify by block number)

Modular Architecture, Modular Design Methodology, Switch Simulation, High Order Language Analysis, Signal Processing, Switching Systems

20 ABSTRACT (Continue on reverse side if necessary and identify by block number)

A modular design process as applied to signal processing and switching systems is described. Architecture analysis, high order language evaluation and comparison, system simulation, and hardware/software tradeoffs were made in the context of modular design of signal processing and switching systems. The principal motivation for modular system design is lower life cycle cost.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

37269

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TABLE OF CONTENTS

APPENDIX A

ARCHITECTURES STUDIED

A.1 PARALLEL BUS ARCHITECTURE

A.2 SERIAL BUS ARCHITECTURE

A.3 MATRIX SWITCH ARCHITECTURE

APPENDIX A-1

PARALLEL BUS ARCHITECTURE

A.1 PARALLEL BUS ARCHITECTURE DESCRIPTION

A.1.1 General

This section describes a parallel non-dedicated bus architecture. The bus chosen resembles the PDP-11 Unibus* in that it is a subset of it. There is a hierarchy of control defined within this architecture.

The first level in this hierarchy is the control mechanism for data transfer on the bus, which is specified in great detail in the remaining section. In general terms, the modules act in master-slave states during a data transfer. All data transfers within the bus are fully interlocked non-synchronous types. The interlocked non-synchronous facility provides compatibility between slow and fast access modules. Electrical daisy chaining between the bus acquisition logic of the Bus Exchange Units (BEU) resolves contention. The general nature of this type of transfer facility allows for one module to continue to hold the bus indefinitely. To prevent this from being a problem, single word transfers should be used by the modules. No penalty in bus bandwidth is paid because the bus acquisition sequence has two states "next master" and "current master". By placing the limit of single word transfers on the modules the highest priority modules is guaranteed 50% of the available bus transfers. A hazard still exists when using slow access modules on the bus. Analysis must be performed to insure that the bus is not held by any module long enough to prevent use by the higher transfer rate modules. To prevent failures in the "bus acquisition logic" from holding the bus too long, a time-out has been defined on the next master/current master states.

The second level of control is that of the algorithms within the modules. The signal processing examples studied in this project may be pipelined so that centralized timing and control is adequate. This type of control is quite desirable since it simplifies the design and analysis of the control facility. This type of control facility is described in further detail later in the text. The study of the switching system example indicated a desire for distributed timing and either centralized or decentralized control. The interrupt port on the BEU allows entry to the controller from any module in the system. This provides for existence of either of the two control types. When using decentralized timing with centralized control, the contention and queueing within the control module is complicated and requires careful analysis.

A.1.2 Bus Architecture Use

The use of the architecture for the algorithm or functional level modules allows for a low transfer rate of data and control on the bus. During the module partitioning phase of the design, trade-offs can be made between demands placed on the module control and data rates on the bus. Two main factors need to be considered during this partitioning. The first is to try to partition modules to minimize the requirements placed on common facilities such as a controller or a common data memory. The second is to try to minimize the rate of data flowing from one module to the next. These requirements are based on trying

*Unibus is a trademark of Digital Equipment Corp.

to reduce the bus bandwidth used by each module and to simplify the control. This does not always yield a minimum hardware solution; and if it is determined that the bus is lightly loaded, then common facilities can be used more.

One hazard exists in using more common facilities. The greater dependence one module places on another module or facility, the harder it is to re-use the module in another application or to change the module later in the life cycle of the equipment.

A.1.3 Functional Description

Figure A.1-1 is a block diagram of the parallel non-dedicated bus architecture as might be applied in modular signal processing. The architecture employs three general functions: Timing and Control, Algorithm Modules, and the Transfer Bus Facility. Section A.1.4 of this document describes the operation and use of this architecture.

The following describes the three general functions:

A. Timing and Control (defined as an example for use with the signal processing system)
Real time drives a time base which directs a control CPU (via interrupts) to initiate various computations within the modules. The computations are supported all or in part by modular algorithm units.

B. Algorithm Modules
All module interfaces include Interrupt, Address, Status, Control, and Data. Bus Modules are separately interrupted via the BEU by the control processor to initiate computation and transfers. This action defines the method of hardware communication with software. A second type of communications is two software modules existing within the same computer. If the module interface is still defined as an interrupt vector address, then the module is called by branching to that vector address. Many sorts of multitask executives may be defined within a single computer but will not be addressed in this description. In this example, each module acts as a slave during data input and a master during data output.

C. Transfer Bus Facility
The Transfer Bus Facility is described in more detail in Section A.1.4. The Bus Exchange Unit (BEU) is the communication port to the Transfer Bus. It can be defined in a modular sense such that, for the simplest case, a module would connect directly to the Transfer Bus with the BEU only providing isolation, or in the most complex case, the BEU would provide for bus to bus communication with DMA control facilities. The BEU provides the following functions in the general case but can be constructed of a sub-set of these functions:
   1. Data rate buffering
   2. Address translation between busses for interprocessor connections
   3. DMA control
   4. Bus isolation

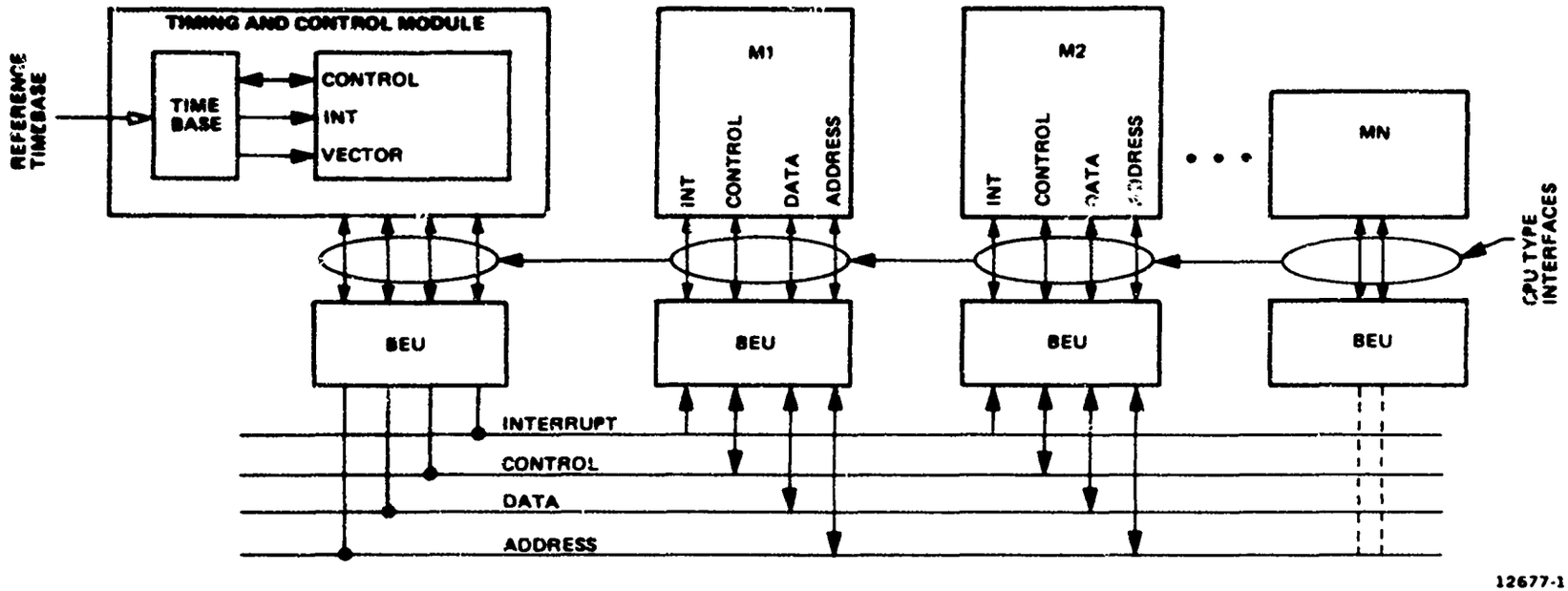


Figure A.1-1. Block Diagram of a Bus Multi-module Architecture.

5. Bus Width Translation
6. Bus Acquisition Logic
7. Interrupt Masking

Figure A.1-2 is a block diagram showing the logic (approx. 30 SSI and MSI ICs) required for any Module which wishes to be a Bus Master (See Section IV) and control data transfer on the bus. This Bus Acquisition Logic would be duplicated, with the DMA control function residing between the two identical blocks, for the most complex case. This represents a total of approx. 90 SSI and MSI ICs.

A.1.4 Principles of Operation

A. Transfer Bus Facilities

The single bus operates with a single path for address, data and control, connecting all modules of the system. The bus operates in a master-slave relationship. At any point in time, at most one device can have control of the bus. Bus communication is inter-locked so that for each control signal issued by the master (controlling) device, there must be a response from the slave (addressed) device in order to complete the transfer. Therefore, communication is independent of response time of the master and slave devices, or the physical bus length. This asynchronous operation precludes the need for transmitting a master clock to all devices, thereby allowing them to operate at their maximum speed.

Priority on the bus is established by wiring position on the bus. The priority structure determines which module is to be granted control of the bus. Every module which is capable of becoming bus master is assigned a priority. When two or more modules request simultaneous use of the bus, the highest-priority requestor is granted control.

The request and granting of bus mastership is performed on an independent set of bus-acquisition lines. While one device is using the bus, the next bus master can be assigned. This overlap allows successive data transfers by different bus master devices to occur with minimum delay.

1. Bus Acquisition

Before a device can proceed with a data transfer, it must become bus master. To become next master, a device must request control at the bus, be granted next-master status, and wait for the current master (if any) to relinquish control. In general, once a device has taken control, it may remain master for as many transfers as desired but in the signal processor case, it is felt that single transfer will be adequate since no overhead is required to re-acquire bus mastership. This allows the highest priority device only 50% of the bus bandwidth.

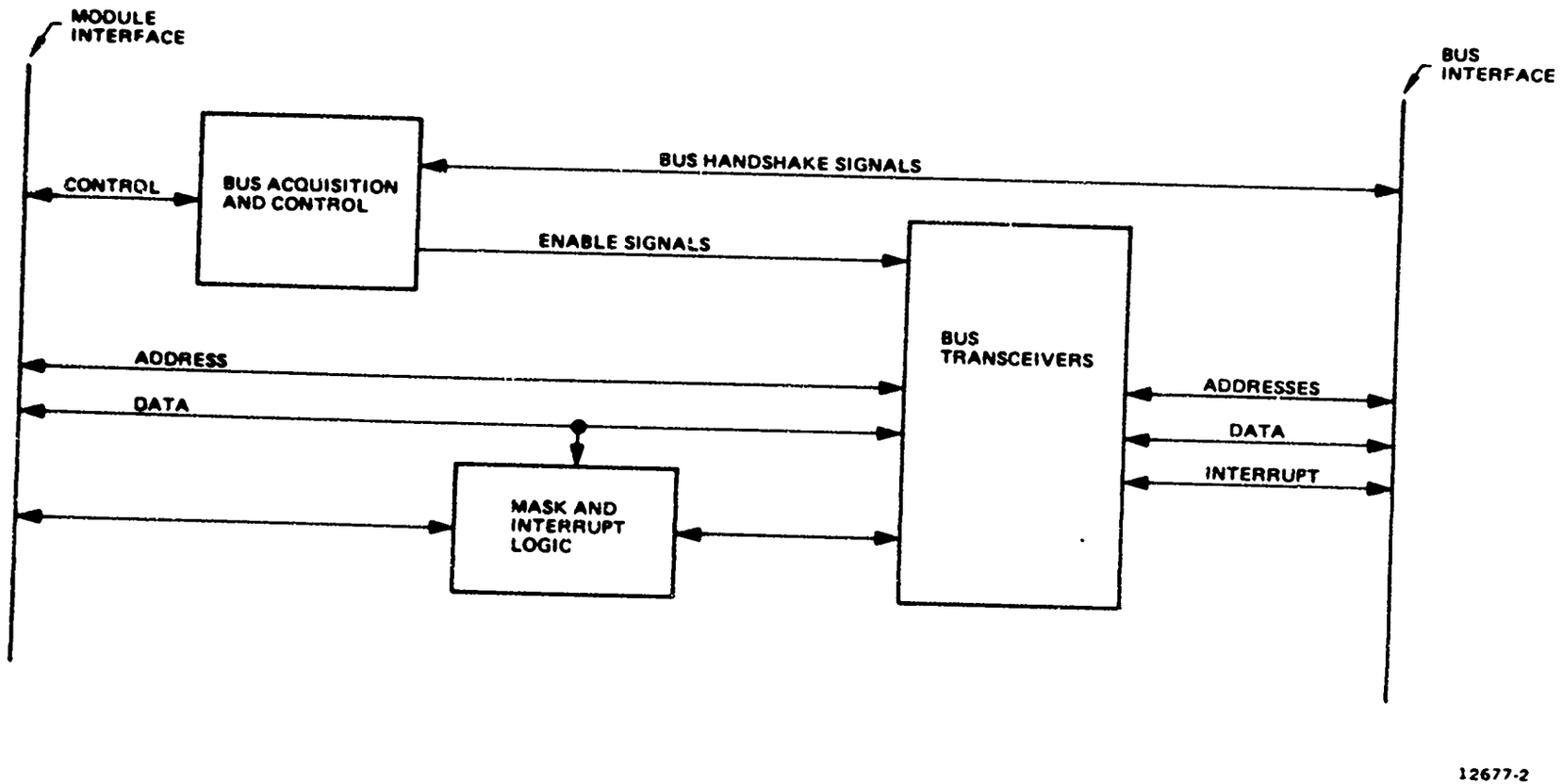


Figure A.1-2. Block Diagram of Bus Master Interface Logic.

The sequence of becoming a bus master can best be understood by referencing Figures A.1-3, A.1-4 and A.1-5. Figure A.1-3 shows the module-to-bus interface with the signals used for bus access. Figure A.1-4 shows a typical bus wiring arrangement to illustrate the priority concept of a module established by electrical chaining. Figure A.1-5 is a state diagram that can be referenced in the following description of the bus acquisition sequence.

The following signals are used for bus acquisition and defined as follows:

a. BRQF - The Bus Request Line is asserted by a module to request bus mastership. A module may assert BRQF any time the seize line (BSZF) is high.

b. BGIF - A chain of Bus-Grant lines provide priority control over which device is to become the next master. The highest priority device receives BGØF which is identical to BRQF, (see Figure 4).

c. BSZF - When a device has accepted a grant, it becomes next master and notifies all other devices by asserting the bus seize line. These other devices inhibit their request until BSZF returns high.

d. BBSYF - The bus busy line is asserted by a device after it has been granted next-master status and when BBSYF is high. It is released by the device after it has transferred its data.

The signal sequence is shown by the state diagram of Figure A.1-5 and proceeds as follows, starting from State 0.

a. Module I needs control of the bus. When BSZF is high, it asserts BRQF (State 1).

b. BRQF is passed through all modules of higher priority and is received at Module I as a low to high transition on BGIF signaling bus grant (State 2).

c. Since Module I is requesting, it keeps its Bus-grant output (BGi+1F) high, blocking the grant from lower priority modules.

d. Module I is now the Next Master. It waits for the Current Master (if any) to set BBSYF high. Module I then asserts BBSYF and clears BRQF.

e. Module I waits for BGiF to return high. It then releases BSZF to enable other Modules to become Next Master. (State 3).

f. When Module i has completed its transfer of data, it sets BBSYF high to allow another module to become Bus Master (State 4 to idle).

2. Data Transfer Operation

As was stated earlier, before a module may proceed with a data transfer, it must become Bus Master. Once it has been granted bus mastership, the module takes control of the control lines. The following signals are used in a data transfer sequence.

a. AB15F - AB00F - The 16 address lines are used by the master device to select the slave with which it will communicate.

b. DB15F - DB00F - The 16 data lines are used to transfer data (in parallel) between the master and the slave.

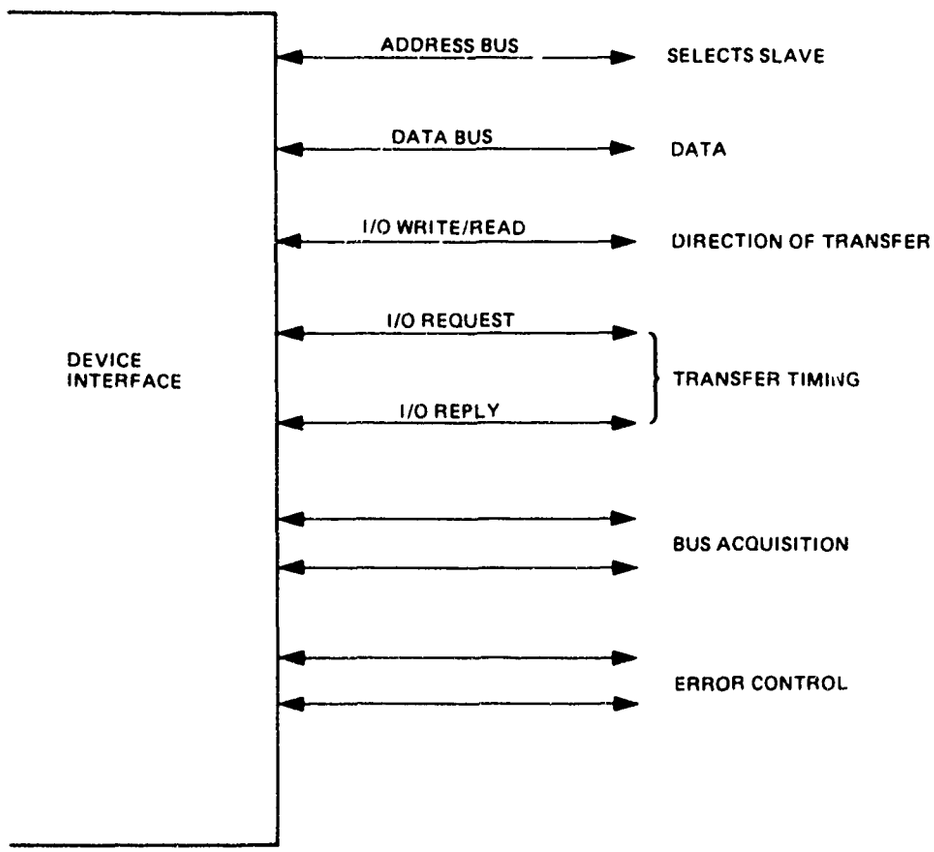


Figure A.1-3. Device Data Transfer Interface Signals.

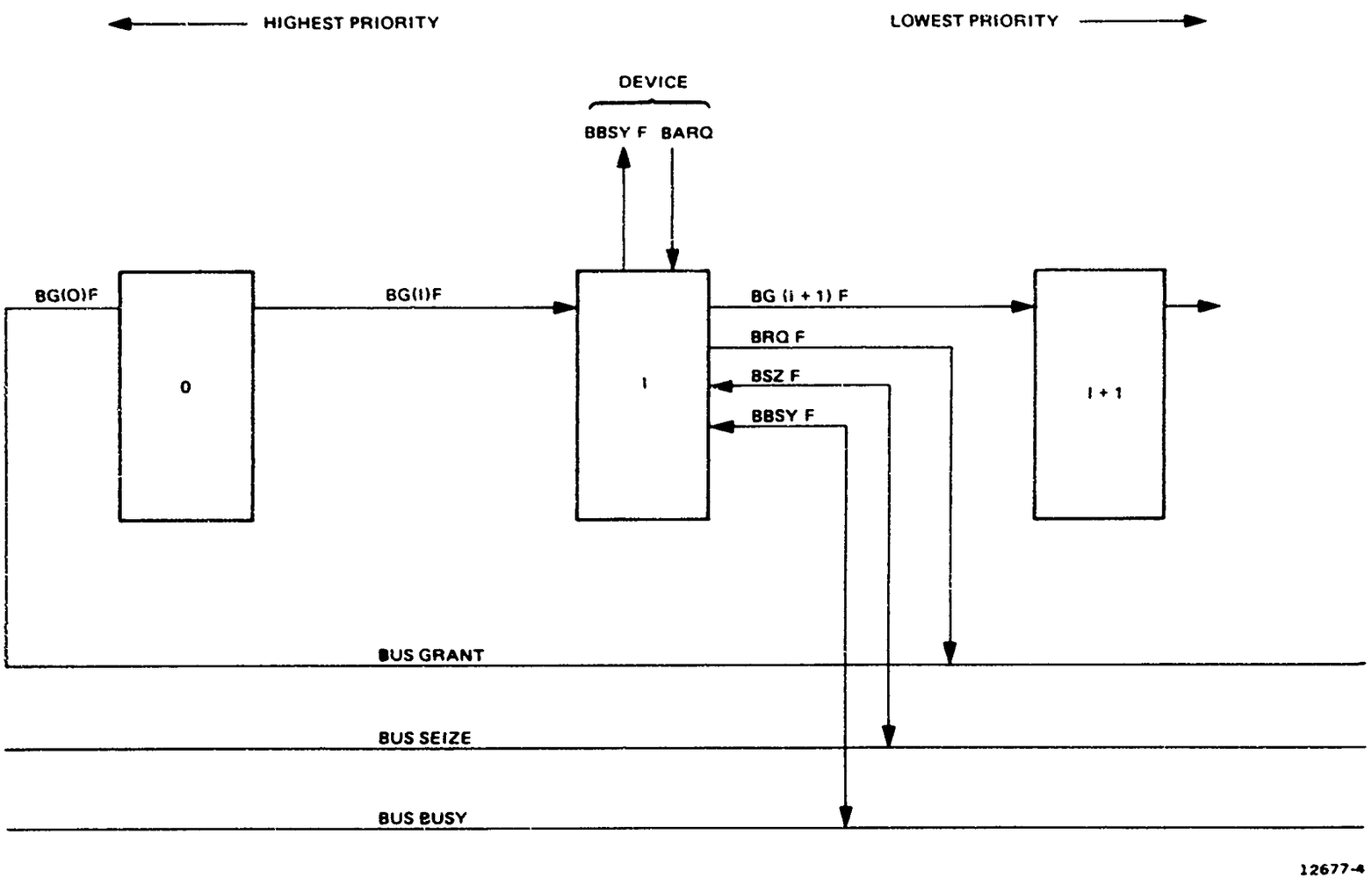


Figure A.1-4. Device Acquisition Wiring.

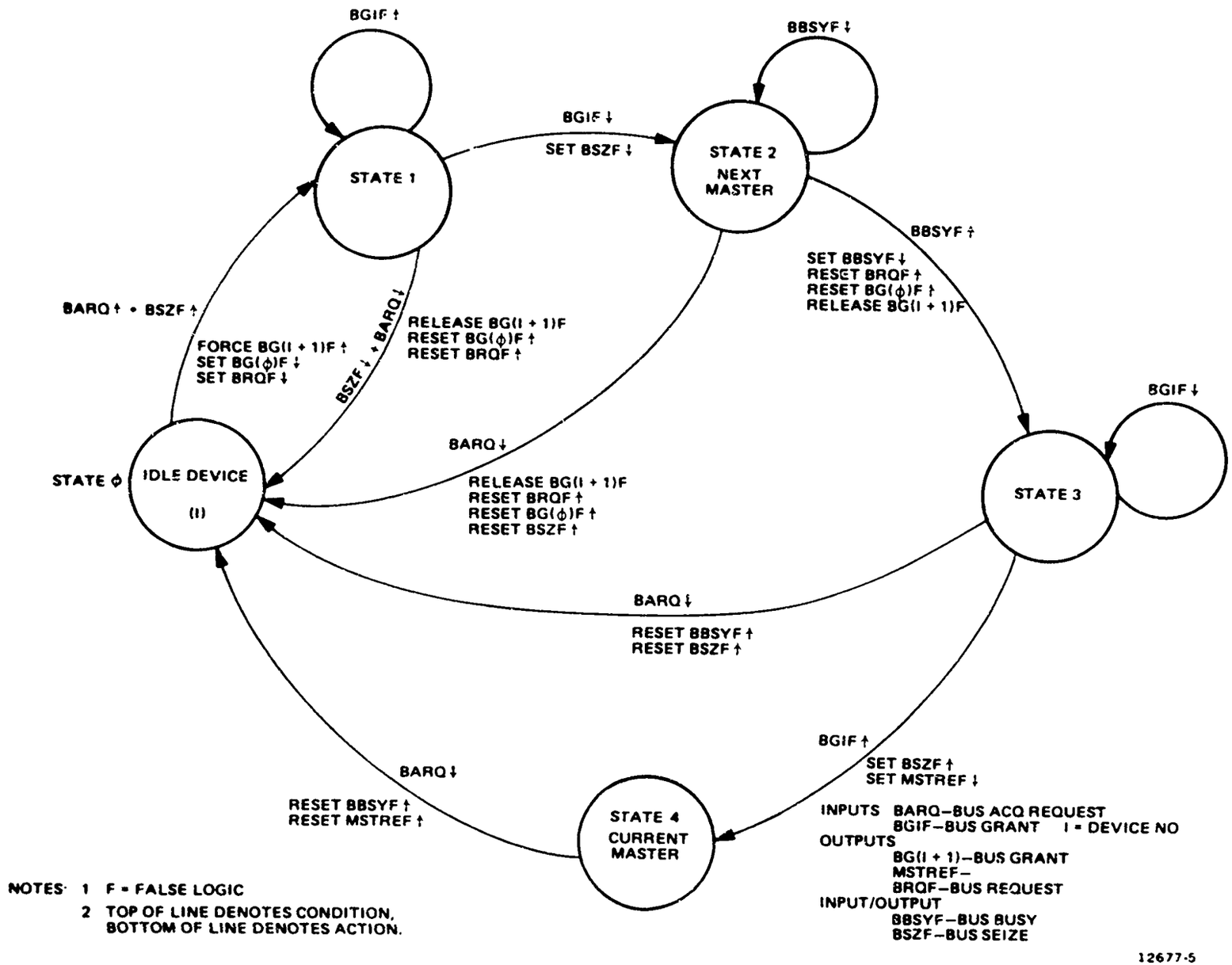


Figure A.1-5. A Low Contention Bus Acquisition Scheme for Single Bus Architecture.

c. IOWRF - The state of IOWRF determines the direction of data transfer, which is with reference to the master: A write (IOWRF low) is from Master to slave; A read is from slave to master.

d. IORQF - The I/O request line is asserted by the master to indicate to the slave that it should proceed with a data transfer.

e. IORPF - The I/O reply line is asserted by the addressed slave in response to the Master's request. Its assertion indicates that the slave has captured the data or gated the read-data to the bus.

With the above definitions of the signal lines, the timing sequence of a read and a write sequence can be examined. Figure A.1-6 shows the control line sequence for a write operation, and Figure A.1-7 shows the control line sequence of the read operation. The times shown are for an assumed bus propagation delay. The arrows between events denote a cause effect relationship and imply a non-negative time delay.

B. System Timing Control

The timing and control of a single bus architecture has an unlimited number of variations possible. The one being investigated was chosen to simplify the control. It takes advantage of the fixed rate processing inherent in signal processing tasks. Signal processing also tends toward a pipeline process which allows for an orderly flow of information from one module to the next. The executive chosen relies on hardware (or programmable hardware) to produce the real time intervals used for task scheduling. Each interval is identified with a unique code or interrupt vector which identifies various tasks.

1. Software Control

The control software is divided into three sections, task scheduler, task queuing, and task dispatcher, as shown in Figure A.1-8.

Operation of the control section assumes foreground/background processing. The background processing (task dispatcher) looks at the task queue and generates control interrupts to the various system modules, informing them to start processing tasks. These interrupts also cause the transfer of data from one module to the next. If timing constraints within a module are severe, a double buffer concept will be used for passing data between modules. The control section will normally specify memory addresses for use by a module. Using the double buffer concept, this address would only be a starting location, with the module using an index to specify addressing from the starting location.

The foreground processing (tasking scheduler) is started by an interrupt from the timing section. This interrupt is interpreted, and the defined tasks are placed in the task queue. Upon completion of this sequence, background processing is resumed.

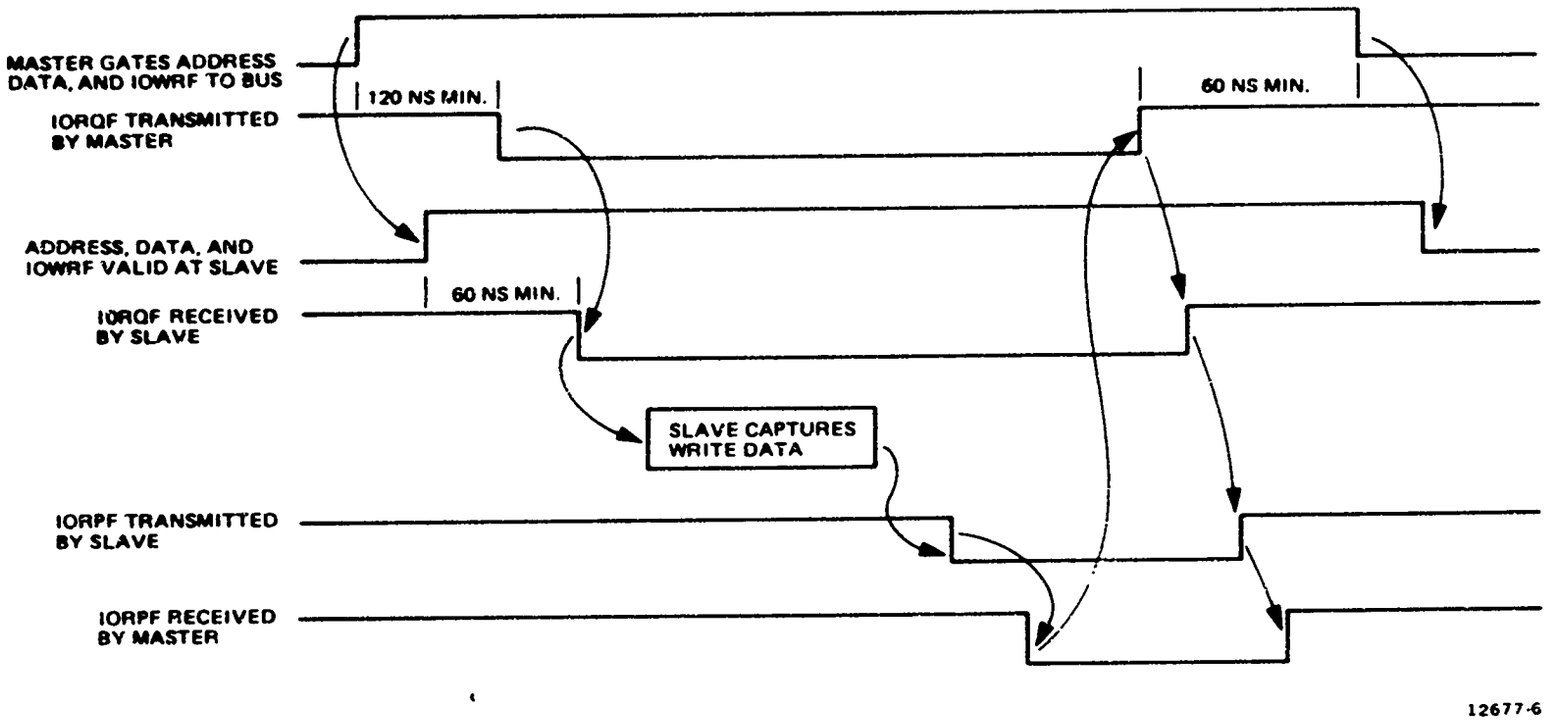


Figure A.1-6. Control Line Sequence for a Write Operation.

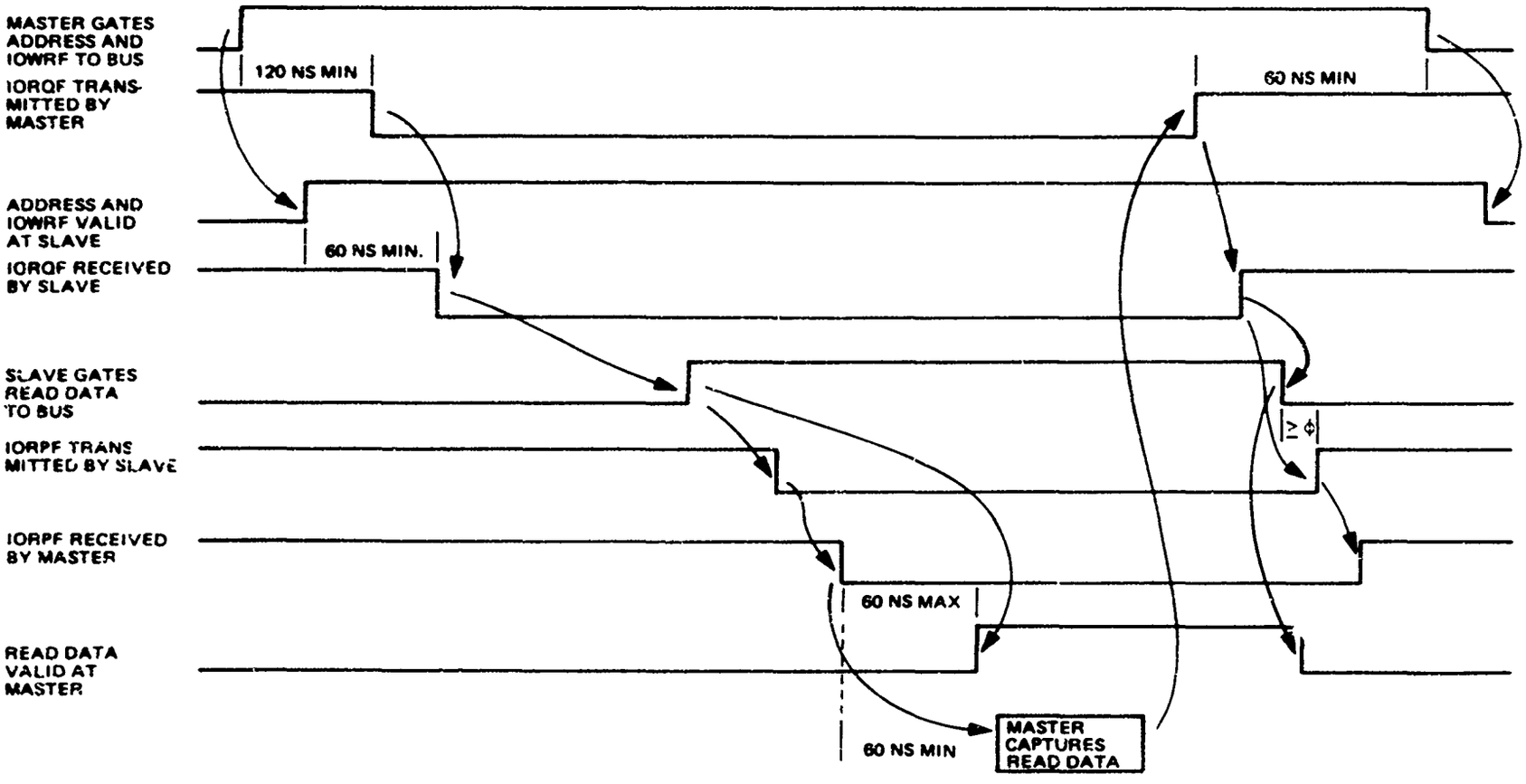


Figure A.1-7. Control Line Sequence for a Read Operation.

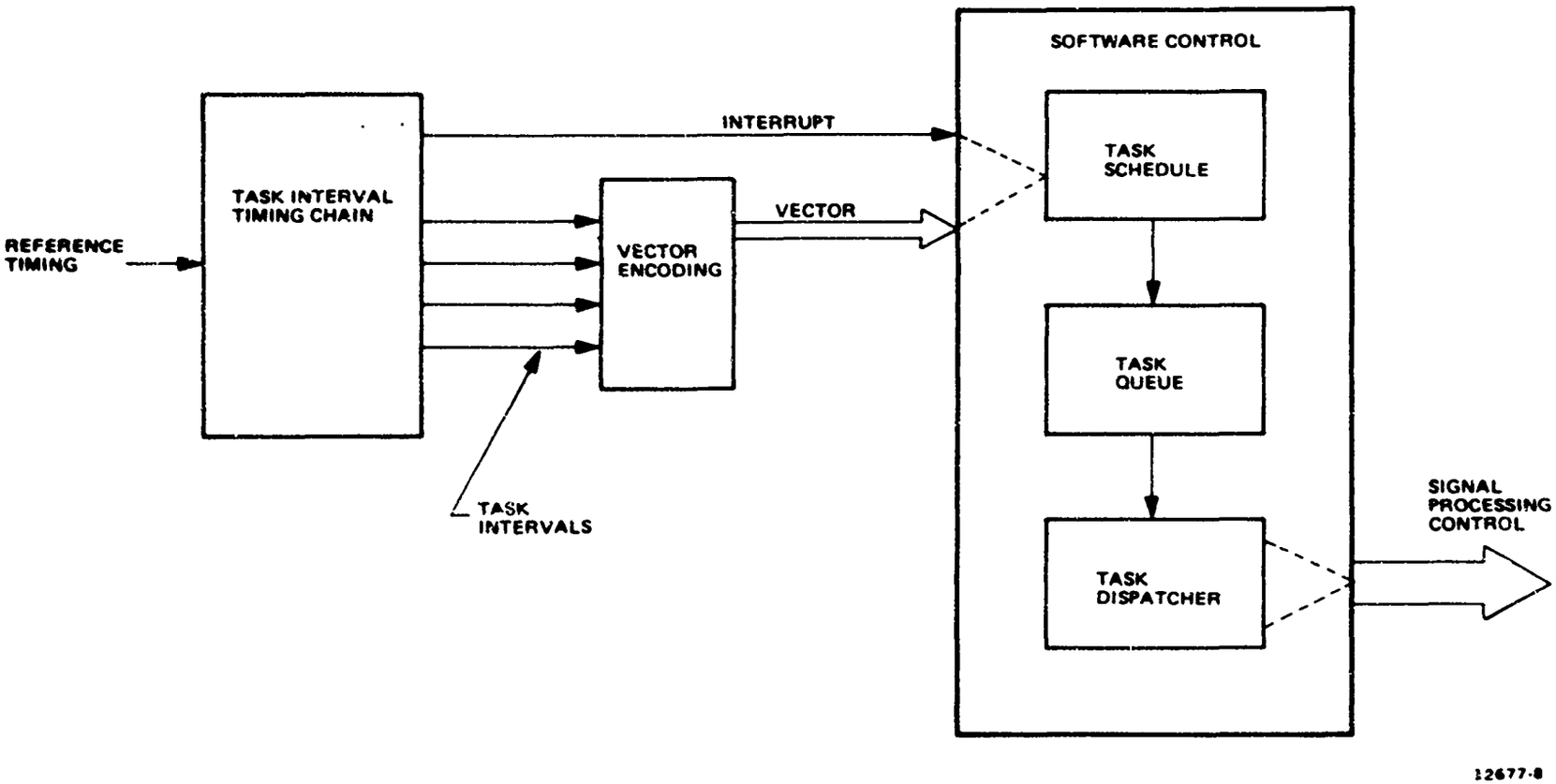


Figure A.1-8. Timing and Control.

2. System Interrupts

There are two types of interrupts within the system. The first is the interrupt group used to drive the Control Software as described above, and the second are those interrupts generated by the Control Software to inform the system modules to start a task. The latter interrupts appear as vectors, with the data interface used to specify the vector address. The control module must first cycle through the Bus Acquisition sequence before issuing an interrupt. The bus interface logic associated with each hardware module monitors vector addresses. When an assigned address is detected, the interrupt signal is passed to the appropriate module. Upon receiving the interrupt, the module will acknowledge the interrupt and the sequence is complete, allowing the control module to release the Bus Acquisition Logic. Under special conditions, a single vector address can be assigned to more than one module. In this case one of the modules acknowledges the interrupt and the rest of the modules must unconditionally respond to it.

C. Error Monitoring

A certain amount of error monitoring equipment has been specified as part of this system. Two classes exist, the first is error detection logic, and the second is error recovery logic.

1. Error Detection - The first level of error detection is by time-out on a data transaction. The Bus Acquisition line is monitored in the timing and control logic, and if the time that any given module is connected to the bus exceeds the time-out, a bus error is declared. A second type of error detection involves the use of Tri-state transceivers with parity generation circuits included. This allows generating the parity across both addresses and data by a master device. The control unit then monitors data and address on the line receivers and declares a bus error if there is a mismatch.
2. Error Recovery - Two levels of error recovery are defined. The first is a Bus Error Interrupt (BEIF) sent to all modules if a time-out occurs. This is used to clear a master from the bus if a failure occurs.

The second level is used to initialize modules, or isolate modules with failed bus drivers. To do this, a switchable power connection needs to be made to the bus driver circuits. A module address would uniquely identify the drivers to be turned off and the parity checking circuits would check for a clearing of the failure. This logic would only be on-line and automatic in a system with distributed processing since a failure of a bus module in a non-redundant system causes down time. It is, however, useful to retain this hardware for fault isolation in any single bus systems to minimize down time.

APPENDIX A-2

SERIAL BUS ARCHITECTURE

A.2 SERIAL BUS ARCHITECTURE DESCRIPTION

A.2.1 General

In a serial bus interconnection system, modules are connected using a serial line, or lines, for the transfer of data and control messages. These message types may be sent on different lines, or time-division and/or frequency-division multiplexed onto the same line. Multiplexing may also be used to increase the capacity of the lines. The main attraction of a serial bus is that very few lines connecting modules are necessary, which, in a physically dispersed system, will greatly lower costs.

A.2.2 System Components

The system described here is based on the MITRE Project 2290 Proposal (M74-225), a serial system using two CATV coaxial cables connected at one end by repeaters, with modules/devices tapped into both lines (Figure A.2-1). A number of busses are frequency multiplexed onto the cable, with each bus (frequency band) time division multiplexed. One bus, the "Control Channel", is for control functions. In addition one or more frequency bands are used for data transfer. The control partitioning is such that each module addition does not require more hardware than is necessary to interface the module to the bus, unless more bandwidth is required. In that case, another frequency band could be added with the addition of a repeater for that frequency band. The elements of the system are further described below.

A. Repeaters (Figure A.2-2)
   One repeater is required for each frequency band used. Each repeater includes a modem and storage needed to synchronize input and output messages. Synchronization is achieved by writing into one buffer while outputting another buffer to the outbound channel. A third buffer is idle at this time, but will output at the next frame time, while the one that previously output will now receive from the inbound cable. The control channel repeater also contains Master Clock Timing and the frame sync generator. Timing at each Module Interface Unit (MIU) is derived from a single source, e.g. derived from the bi-phase PSK signal transmitted by the repeater for each bus, and generates the Master Clock Timing.

B. Amplifiers
   Amplifiers are required to boost the signal when a set amount of loss has occurred along a path, e.g. 20db. It is proposed that the present wideband distribution amplifiers used in CATV, which have an established high reliability, would be ideal for this application.

C. Taps
   The taps used are simple passive devices, similar to those used in the CATV industry.

D. Module Interface Unit (Figure A.2-3)
   The Module Interface Unit (MIU) contains the modems, logic for control decoding and data handling, and an I/O adapter which interfaces a device/module to the coax cable. This unit is rather complex

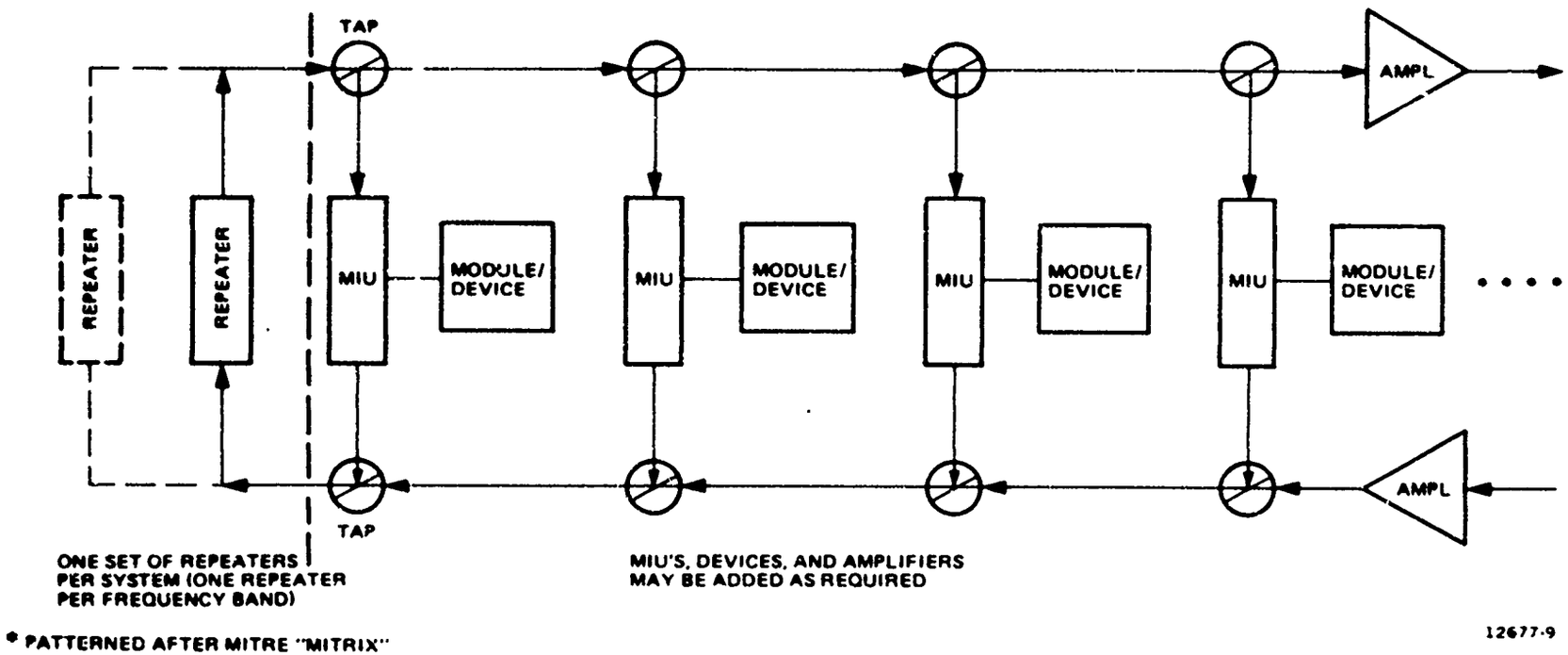


Figure A.2-1. Serial Bus Architecture.

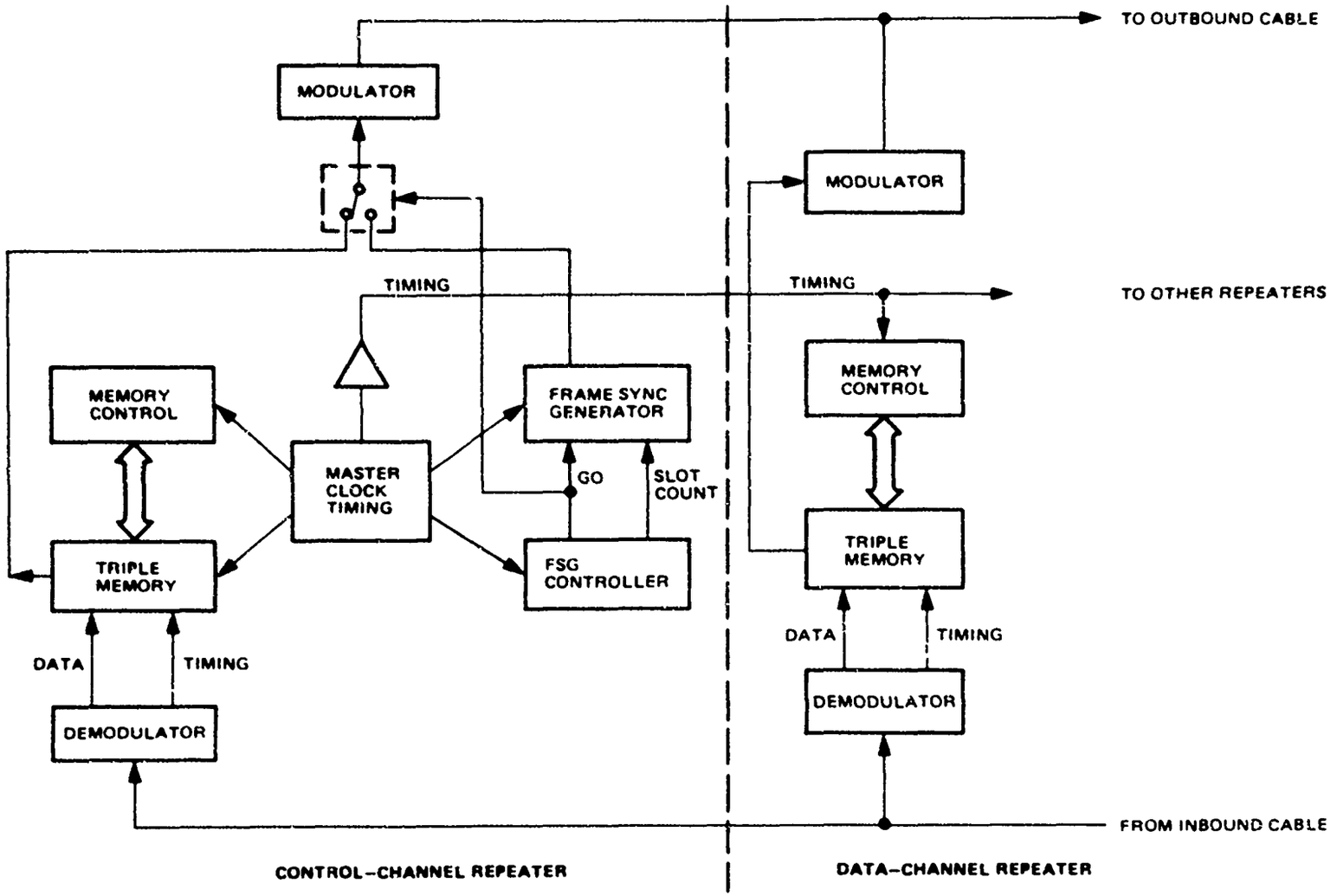


Figure A.2-2. Repeater Block Diagram.

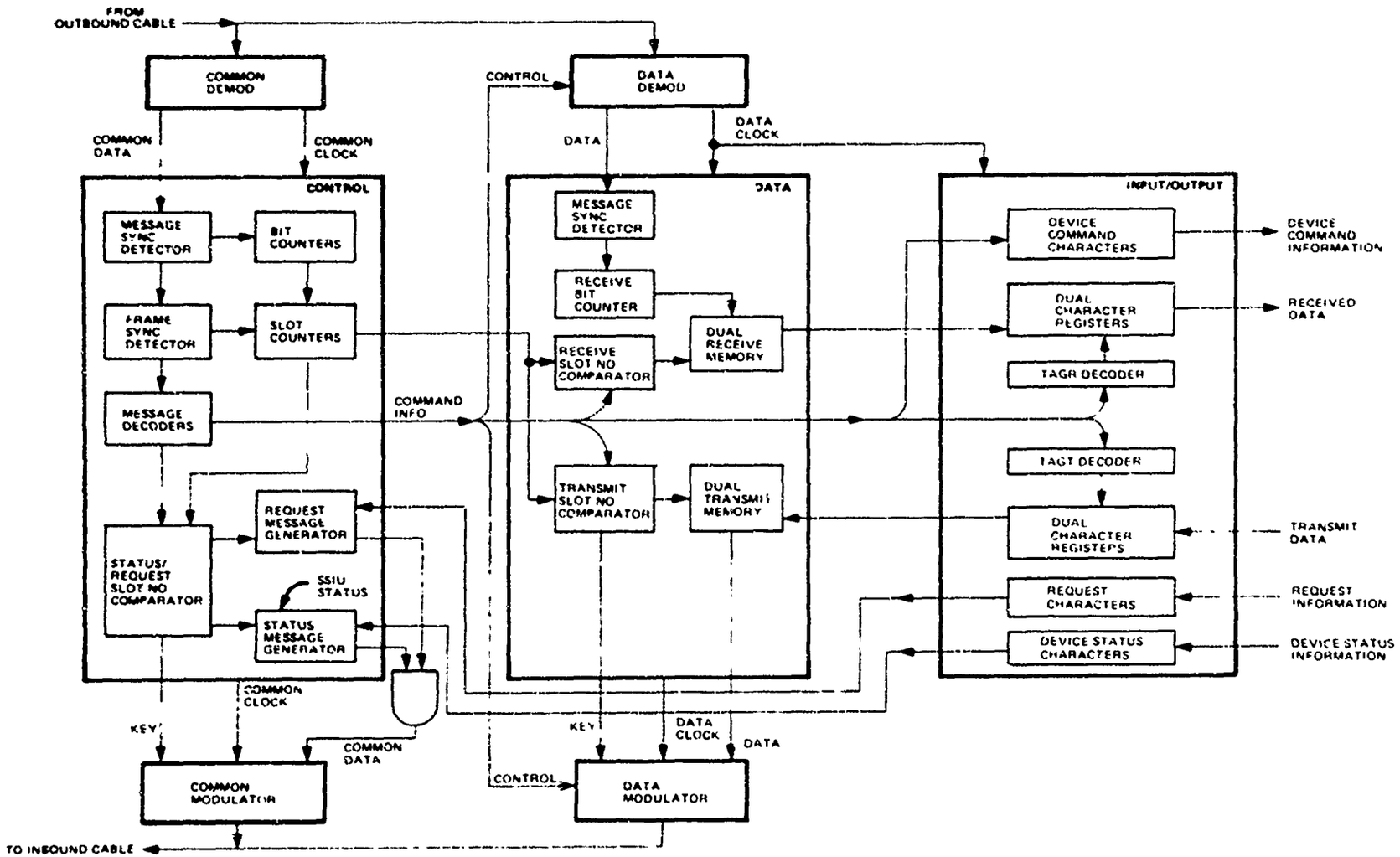


Figure A.2-3. Module Interface Unit Block Diagram.

(over 90 MSI and 30 SSI ICs) and this complexity constitutes part of the cost trade-off with other architectures. The MIU is broken down into three sections: Control, Data, and I/O.

1. Control

   The control section of the MIU, on the left side of Figure A.2-3, includes the control channel modem, counters to keep track of transmit and receive slot numbers and the bit count within each slot. These are synchronized by the frame-sync detector and the message-sync detector, respectively. A series of decoders extract command information from a variety of control messages (Figure A.2-4); this information is then routed to various parts of the MIU, and to the module/device attached to the MIU.

2. Data

   The data section, like the control section, contains a receive bit counter and message-sync detector. Message sync is separate for each data channel to allow for differential delay between the control channel and the data channel. Receive and transmit slot comparators monitor the slot count continuously (counted in the control section) and determine the assigned slot to receive/transmit from a slot assignment command. The data section derives the timing for the I/O section, according to the speed required for receiving and transmitting data. Two dual-memories, each one slot in length, act as a buffer between the I/O and the data modems. Dual memory is needed so that one memory may be connected to the data channel, while the other is connected to the I/O.

3. I/O

   The I/O adapter logic provides two-way interface between the data and control section and the module/device that is connected to the MIU. The main function of the I/O section is to exchange information between the data memories described above and the device.

   Typically, in each direction, a dual character register is used for this task. The interface to the device from the dual registers may be serial or parallel. The receive I/O logic "filters" on message tags (Figure A.2-5) in accordance with commands obtained from the control section of the MIU. Tag "filtering" can be commanded into a disabled mode, wherein tag characters would not be examined. Additionally, referring again to Figure A.2-3, the I/O adapter must provide a link for request-message and device status-message characters to get to the control section and for device command characters to be delivered to the connected device.

E. Bus System Control: Technical Control, Process Control

1. Technical Control examines hardware functions including monitoring, assessment, and control of the network operation relating to bus usage. System components under Technical Control include the bus cable, MIUs, repeaters, I/O interfaces and terminal devices (external control only). Technical Control is performed by built-in equipment in the MIU and a Technical Controller which is connected in the network as one of the modules/devices. The Technical Controller only communicates through the control bus.

2. Process Control receives processing requests, assesses the requirements to execute the job, assigns the necessary devices, and oversees the processing. The Process Control communicates any failures or blockages it encounters to the Technical Control, which may allow process control to reassign equipment or time slots.
3 Technical and process control may be implemented in a single minicomputer with the software for the two sections separated, or by two small minicomputers, each dedicated to one part of the control.

A.2.3 Message Formats (Figures A.2-4 and A.2-5)

These are two major message formats used in the digital system, Data and Control. Each message occupies one time slot (314 bits long) and begin and end with an 8-bit guard band. Each message also contains an 8-bit synchronization sequence.

A. Data [D] Messages

Data messages are defined as those which contain data to be processed by the modules/devices. They will only be transmitted on the data busses. The standard format (top of Figure A.2-4) uses 34 bits for tag and 256 bits for data. Message tags are used to address specific MIUs and/or to identify data content (Figure A.2-5).

B. Control Messages

Control bus command and control message formats use the 290 bits (256 + 34) as illustrated in the remaining entries of Figure A.2-4. Control Messages are broken down into two groups; network control messages and operational control messages. Eight bits are used to determine the type of control message being received.

1. Network Control [N] Messages are used for the technical monitoring assessment and control of the entire bus system.
2. Operational Control [C] Messages provide the means for transmitting mission-oriented control information over the data distribution (control channel only) system. All control transactions directly involving operational data or processes are performed using category C messages.

A.2.4 Control Scenario

To demonstrate the control operation on the bus, the system synchronization and process request sequence is expanded in the paragraphs below. The letter-number combination in parentheses; (N5), (N1), etc.; refer to the message formats of Figure A.2-4.

A. Serial Bus System Synchronization and Slot Acquisition

The repeater for the common bus channel generates eight evenly spaced frame synchronization messages (N5) within each frame (1 frame = 65,536 slots = 1.4 seconds). When system synchronization has been achieved, the system technical controller (STC) issues status identification messages (N4) to all MIUs which assigns each MIU a slot or set of slots on the common bus during which device/module status messages are to be sent to the STC, as well as the tag to use. During the same

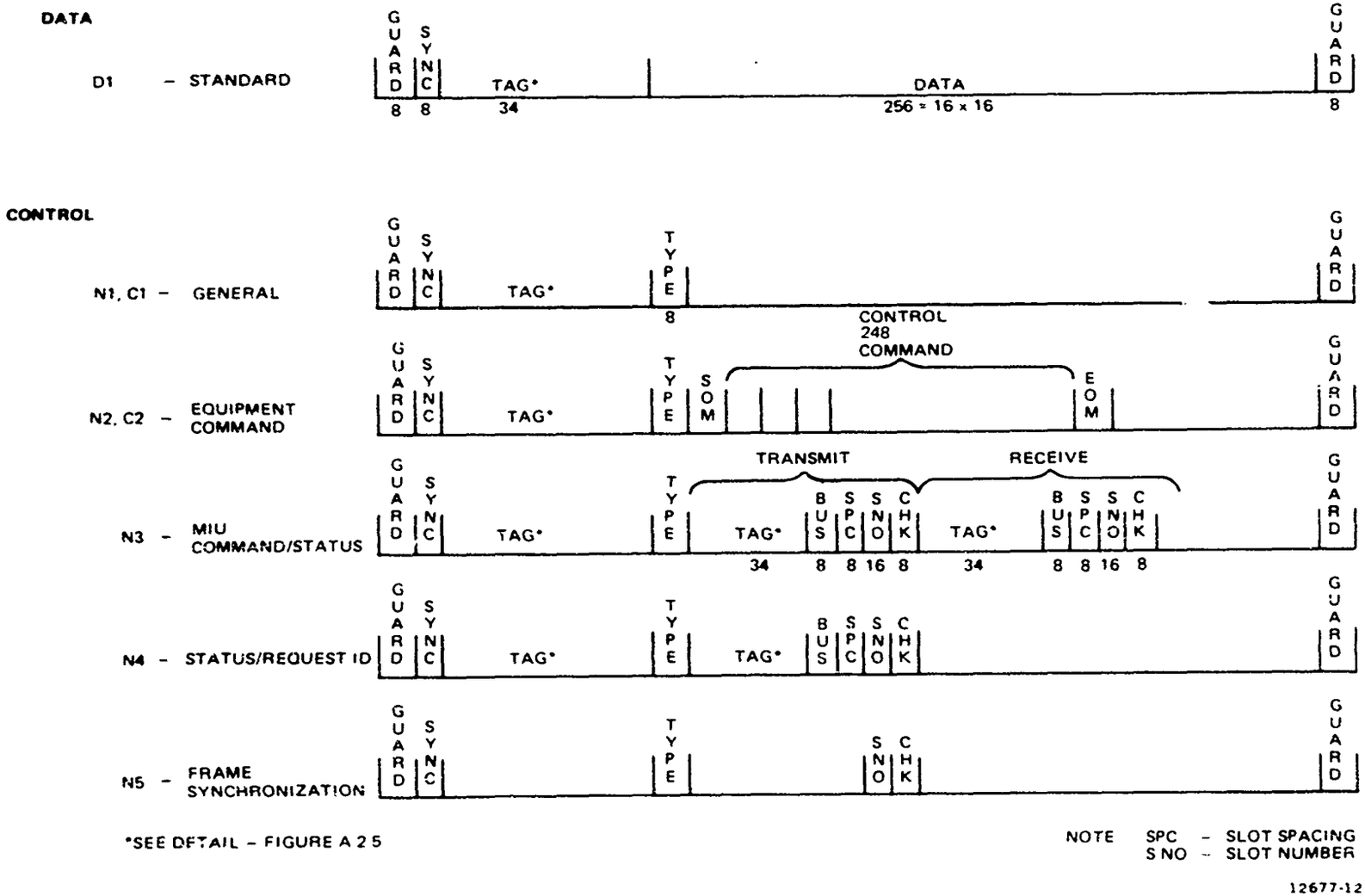
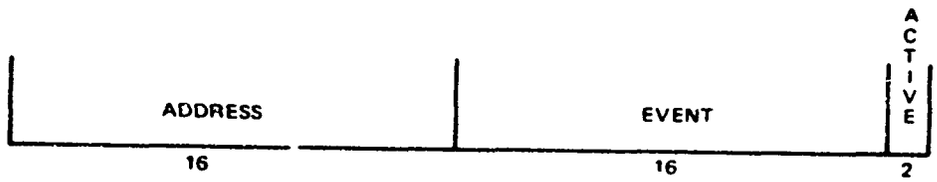


Figure A.2-4. Serial Bus Message Formats.



ACTIVE - INDICATES IF THE TWO PREVIOUS FIELDS CONTAIN INFORMATION, OR SHOULD BE IGNORED

12677-13

Figure A.2-5. Message Tag Format.

frame, the STC issues request identification messages (N4) which assigns MIUs slots and tags to be used when making system or process requests/commands. Each of these ID messages is repeated 2048 times per frame, allowing 2048 MIUs to be updated each frame. If there are less than 2048 MIUs, each MIU could be updated at a proportionally higher rate.

After an MIU receives a status ID message, it begins to search for its assigned slot. When the transmit slot count equals the status slot assignment, the MIU transmits, on the common channel, the status message(s) indicating its status (N1) and the status of the device/module (N1). After all MIUs have transmitted their messages and the STC has examined them, the STC transmits the system status (N1) to the system process controller (SPC). At this point system start-up is complete. The SPC may then accept a process request from an MIU (C1) via the common bus transmitted in the MIUs assigned request slot. When the SPC accepts a process request, it analyzes the request and determines (looks up) the equipment required and the information handling requirements. The SPC may then transmit the requirements to the STC for allocation of available resources or have delegated to itself control of a specific subset of the modules making up the system.

The controlling unit assigns a bus, slot number, slot spacing, and tag information (N3) to each required MIU for proper data flow, and the SPC transmits any processing parameters or operational controls needed by the selected units (C2). During processing, the SPC examines the operation of both modules and MIUs to verify proper operation and may reassign resources to fine tune processing or data handling capabilities.

A.2.5 Conclusions

The system proposed in the paper, while allowing for easy module additions and inexpensive wiring costs, may not be ideal for a modular signal processing system because of the high cost of the MIU, and the delay in communications between modules inherent in the repeaters. As presently implemented, this delay of two message slots (approximately 43 μsec) would severely impact real-time processing. It should be noted that this delay would not interfere with a switching system's operation. A decrease in the slot length would decrease this delay, but increase the overhead-data ratio. It is possible to remove the slot storage in the repeaters in favor of a 16-bit FIFO. This would lower the delay to an average of 8 bits.

APPENDIX A-3

MATRIX SWITCH ARCHITECTURE

A.3 MATRIX SWITCH ARCHITECTURE DESCRIPTION

A.3.1 General

The matrix switch architecture (Figure A.3-1) is similar in concept to the C.MMP multi-mini-processor computer system designed at Carnegie-Mellon University. The difference is that only one matrix switch is utilized rather than two, i.e., C.MMP utilized one matrix switch between memories and computers and a second one between device controllers and computers. In this architecture, the modules, whether it be memory, device controller or computer are all connected to one matrix switch. Thus, each module has the capability to seek connection to another module without use of a third party, i.e., a Device Module can connect directly to a Memory Module instead of to a Computer Module which in turn connects to a Memory Module. The matrix switch itself provides the access logic control, contention resolving, release control, and line status. Connection between modules is a virtual path through the Matrix Switch Module. Each module interfaces with the Matrix Switch Module via a Matrix Interface Controller (MIC). The MIC (and the Matrix Switch) consists of a micro-computer, dedicated ROM and RAM, and an I/O interface with serial to parallel conversion.

A.3.2 Implementation

A. Matrix Switch

1. Hardware Description

The Matrix Switch Module consists of a Matrix Switch controlled by a processor composed of micro-computer elements configured to best provide the circuit switching service of the Matrix Switch Architecture. A block diagram of the Matrix Switch Module is shown in Figure A.3-2. Control lines are connected to the processor and full duplex data lines are connected to the Matrix Switch. Circuits are virtually 'connected' on command of the processor upon request of the calling module. The processor maintains current status on the connectivity and operation of each data line as well as the control lines. Since the matrix switch is the common gateway between all modules in the system, a microprocessor with higher performance and throughput is needed to service all the system module intercommunications with minimum overhead time. A microprocessor composed of INTEL'S 3000 family of two bit slice microcomputer chips is being considered, using a Schottky Bipolar ROM for program storage, and an 85 ns access RAM for data and status. The I/O module control interface provides serial to parallel and parallel to serial conversions, whereas, the interface to the matrix consists of a connect/disconnect line, an eight bit source address and an eight bit destination address. The Matrix Switch may be expected to be composed of tri-state logic elements configured to perform logic connections from one line to another according to source and destination addresses from the matrix switch controller.

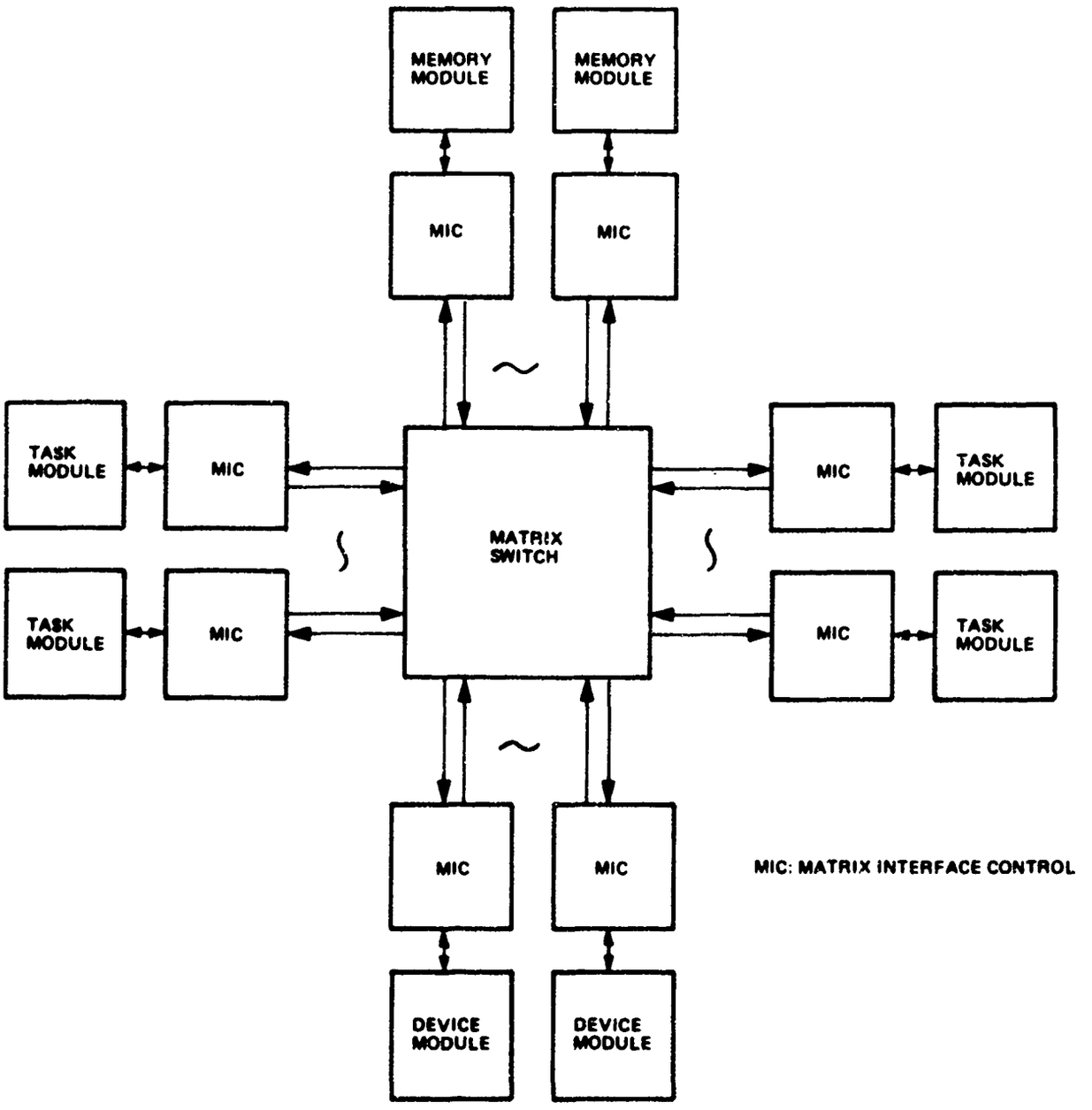


Figure A.3-1. General Matrix Switch Architecture.

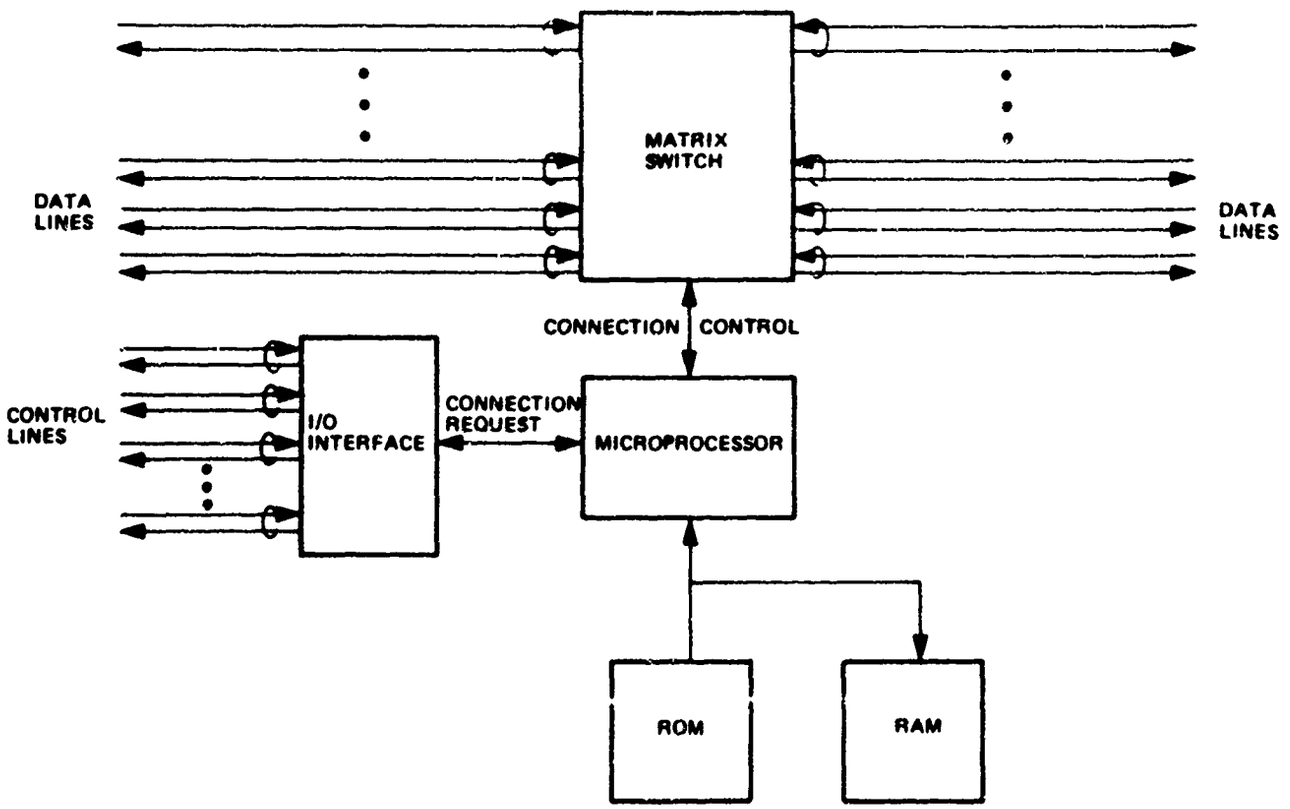


Figure A.3-2. Matrix Switch Module.

2. Principles of Operation

The Matrix Switch module is the focal point of all the system modules, i.e., all modules are connected to it and may only communicate with another module through it. It functions primarily as a slave device in that it performs its switching functions only on command by the System Modules. Each control circuit consists of two lines providing two way bit serial synchronous transmission. The I/O interface provides serial to parallel (8 bit) assembly, allowing the matrix switch processor to operate on 8-bit assembled control characters, saving the processor from the time consuming task of character assembly. The system modules request connections, and the matrix switch controller obliges if the requested destination is available; otherwise a 'NAK' is returned. Disconnections are made on request by the destination system module when data has been transmitted in its entirety.

The Matrix Switch processor basically provides three functions: (a) Monitor for connect/disconnect, (b) Route and determine circuit availability, and maintain circuit status, and (c) connect circuits via commands to the Matrix Switch. Figure A.3-3 shows the connection/disconnection path logic. Figure A.3-4 shows the steps that occur in a module connection cycle.

2.1 Connect/Disconnect Monitor

When a circuit is not connected, idle (or sync) characters are continuously transmitted on the control line to the Matrix Switch Controller. Idle characters are also transmitted back to the sender. These idle characters provide an operational OK indication on both ends of the control line.

When a connection is desired, the module sends a "connect" character followed by a two character circuit address, and idle (sync) characters. The matrix switch monitors each unconnected circuit for a "connect" character, and when found attempts to establish the connection (paragraph 2.2).

When in the connect condition, the matrix switch monitors the destination control circuit for a disconnect character. Thus, disconnection is controlled by the destination module.

2.2 Route and Status

When a "connect" character is received, the next two characters are assembled and used as the destination circuit address. From this address, its status table is obtained and checked for connectivity. If no connection is indicated, an "ACK" is transmitted back to the requester and both status tables (source and destination) are updated identifying its connection and whether it is the source or destination port. A connect command is then sent to the matrix switch.

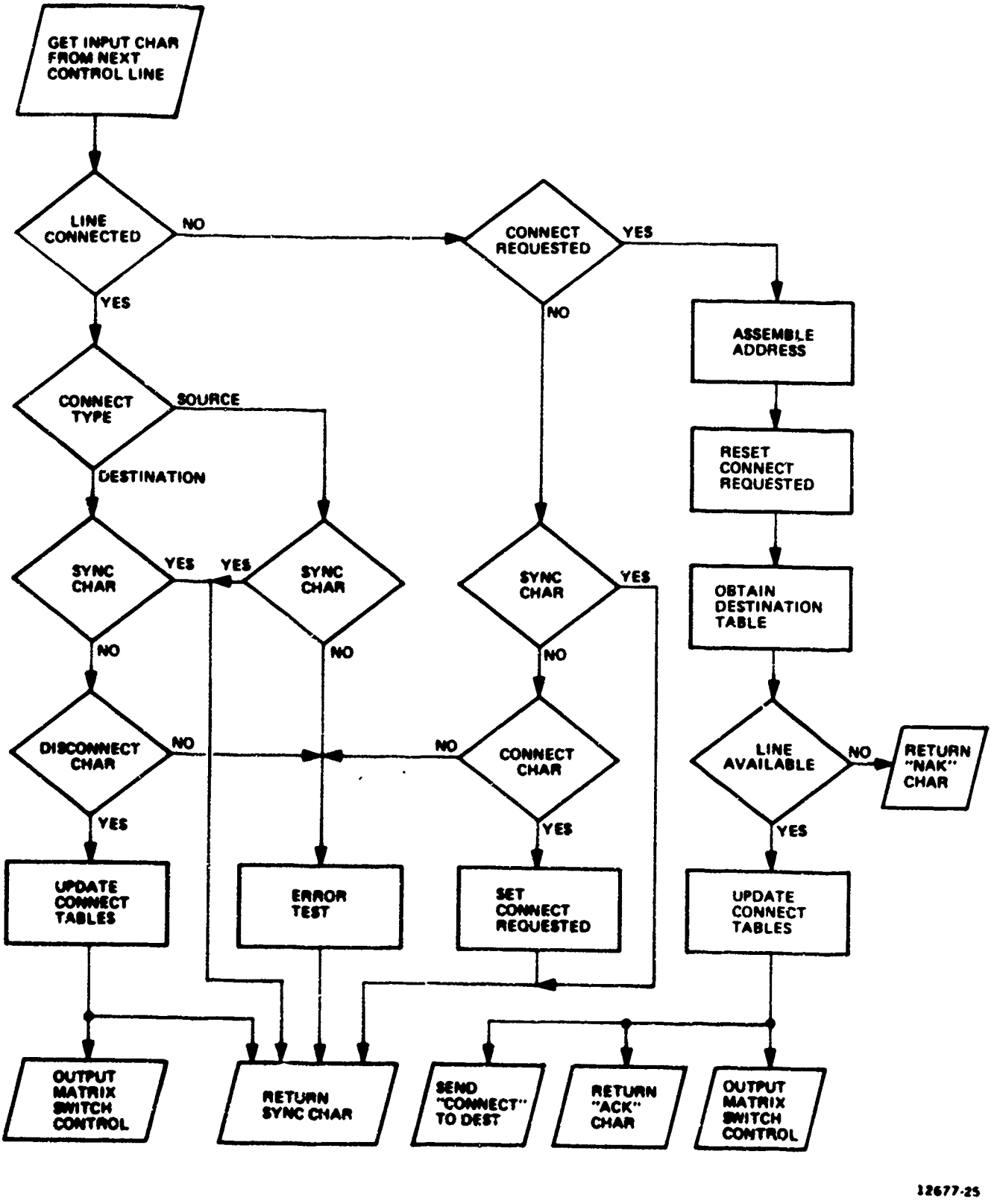


Figure A.3-3. Connection Monitor.

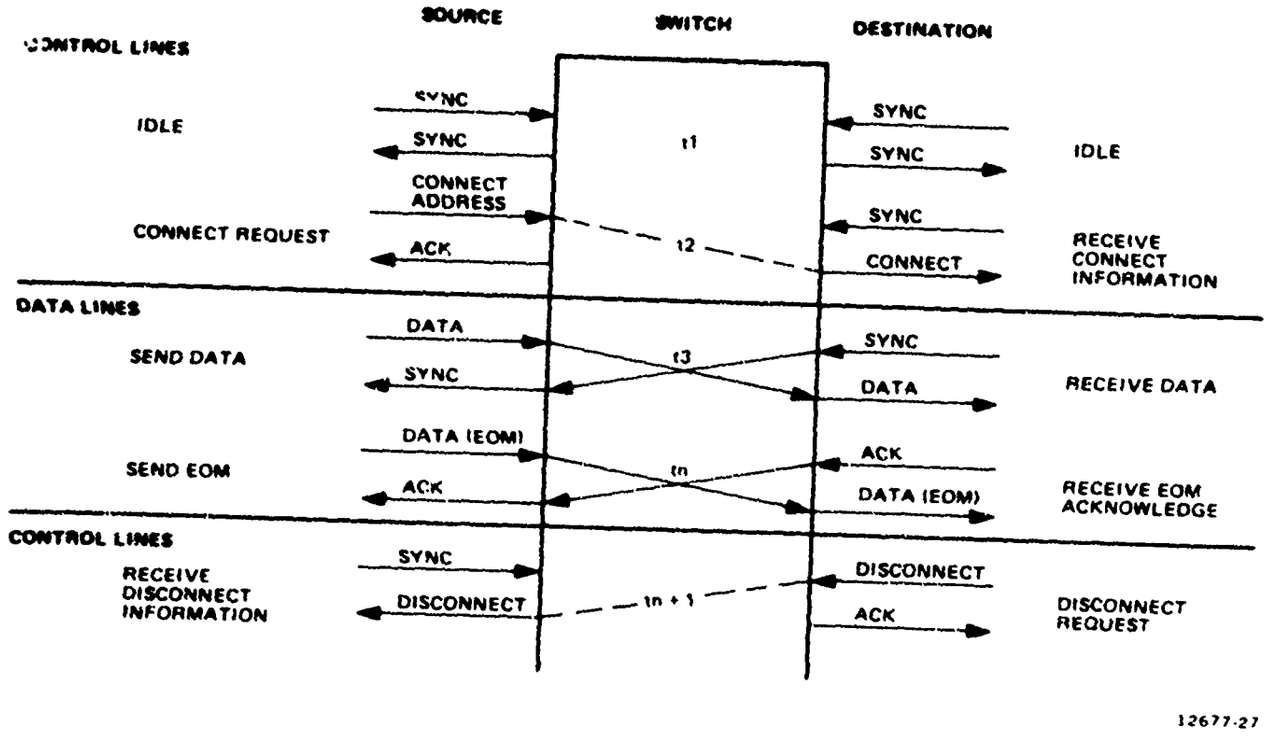


Figure A.3-4. Procedural Dialogue (Module-to-Module Connection).

If the destination circuit is "busy" or "down", the Matrix Switch Controller sends a 'NAK' back to the requester. The requesting module may now continue requesting a connection to the same circuit or request connection to a different circuit.

2.3 Data Transfer

In this mode, the Matrix Switch provides point to point connection between the modules. The Matrix Switch Controller does not have access to the data transmitted; it monitors for disconnect (control command from receiving module) and error conditions.

2.4 Status Table

A sample of the type of information maintained in the status table (for each line) are as follows:

1. Available/Unavailable
2. Circuit Connection Number
3. Circuit Down (No Response)
4. Source/Destination Mode

B. Matrix Interface Control

Similar to the Matrix Switch Controller, the Matrix Interface Control (MIC) consists of a microprocessor with ROM and RAM memory and I/O interface. A lesser capacity microprocessor is used since the MIC is only responsible for one (its own) high speed data and control line. The MIC's primary responsibility is to interface and provide the protocol (described previously in initiating connects and/or disconnects via the Matrix Switch to other MIC's. Three general categories of MIC's are proposed: Algorithm or Task Processor Interface, Device Controller Interface, and Memory Controller Interface. Figure A.3-5a, b, and c show the three general module configurations. The I/O interface to the Matrix Switch is identical for the three configurations. In the memory module configuration (Figure A.3-5b), data is moved in and out of main memory from the RAM via DMA interface. In this type module, the MIC also provides memory management. In the Device Modules (Figure A.3-5c), circuit control and protocol is provided according to the type of device. The data is buffered in the RAM prior to sending to another module (memory or task). In the task module (Figure A.3-5a), the MIC may communicate with the main task processor via DMA or via program controlled processor to processor interface. The task module performs all the functions, tasks, algorithms of the system. It also may be used for scheduling and task queuing functions.

C. Communication Interface

The lowest level of communication interface is at the data/message level where information is originated at one point and received at another point. The data/messages are normally identified with framing characters such as SOH, STX, ETX, EOM, etc., and routing and security data which provide the information needed to process, route, and transmit the data/message to its destination.

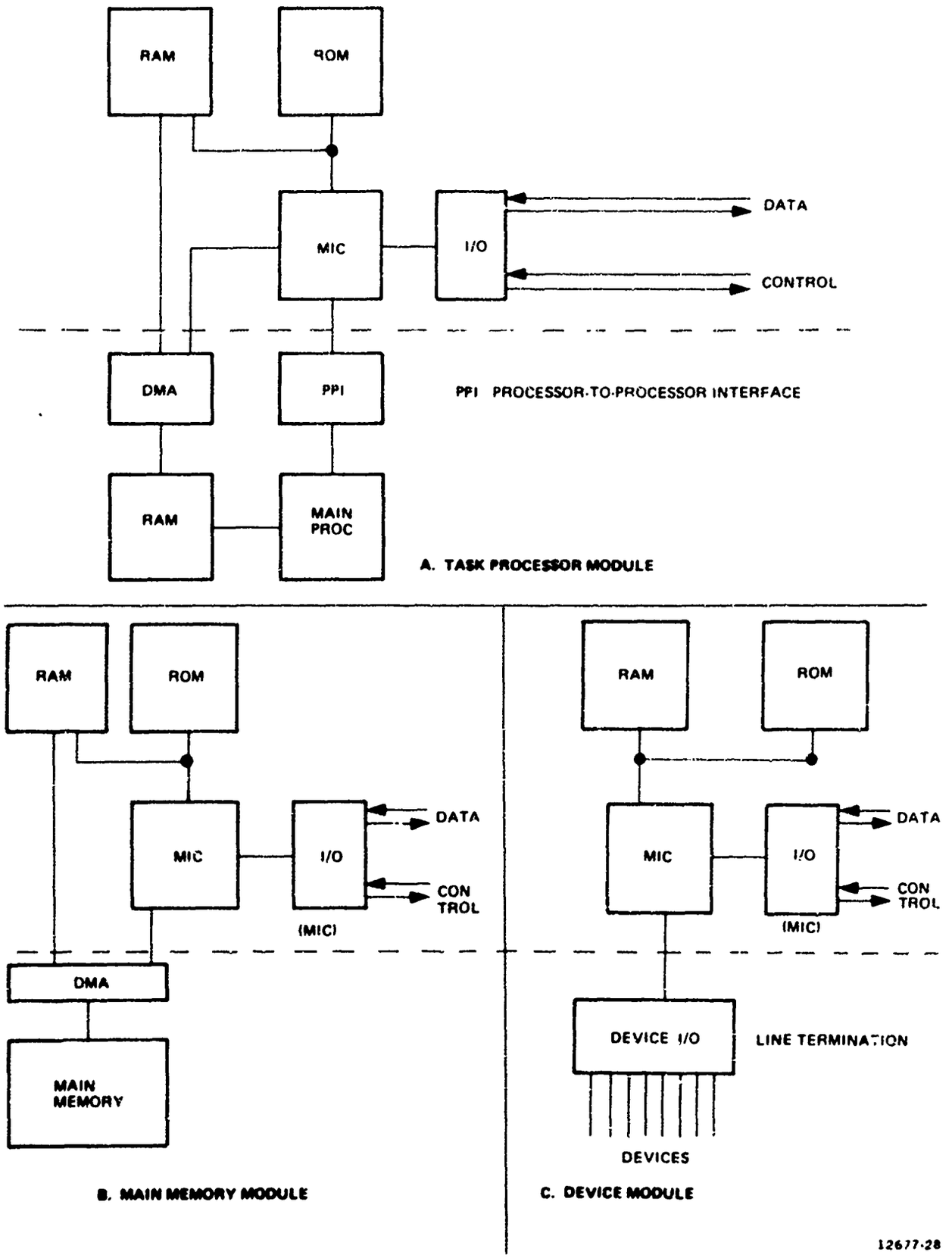


Figure A.3-5. Control Logic Modules.

The next level of communication interface is the device to system communication level (dashed lines in Figure A.3-5c) where unique device protocol must be adhered to. The highest level of communications interface is at the system level where communications are generated to interconnect various components of the system. This is the communication level that is unique to the system architecture. This level is further separated into two sublevels, the module to module communication interface and the module to matrix switch control interface, as described below.

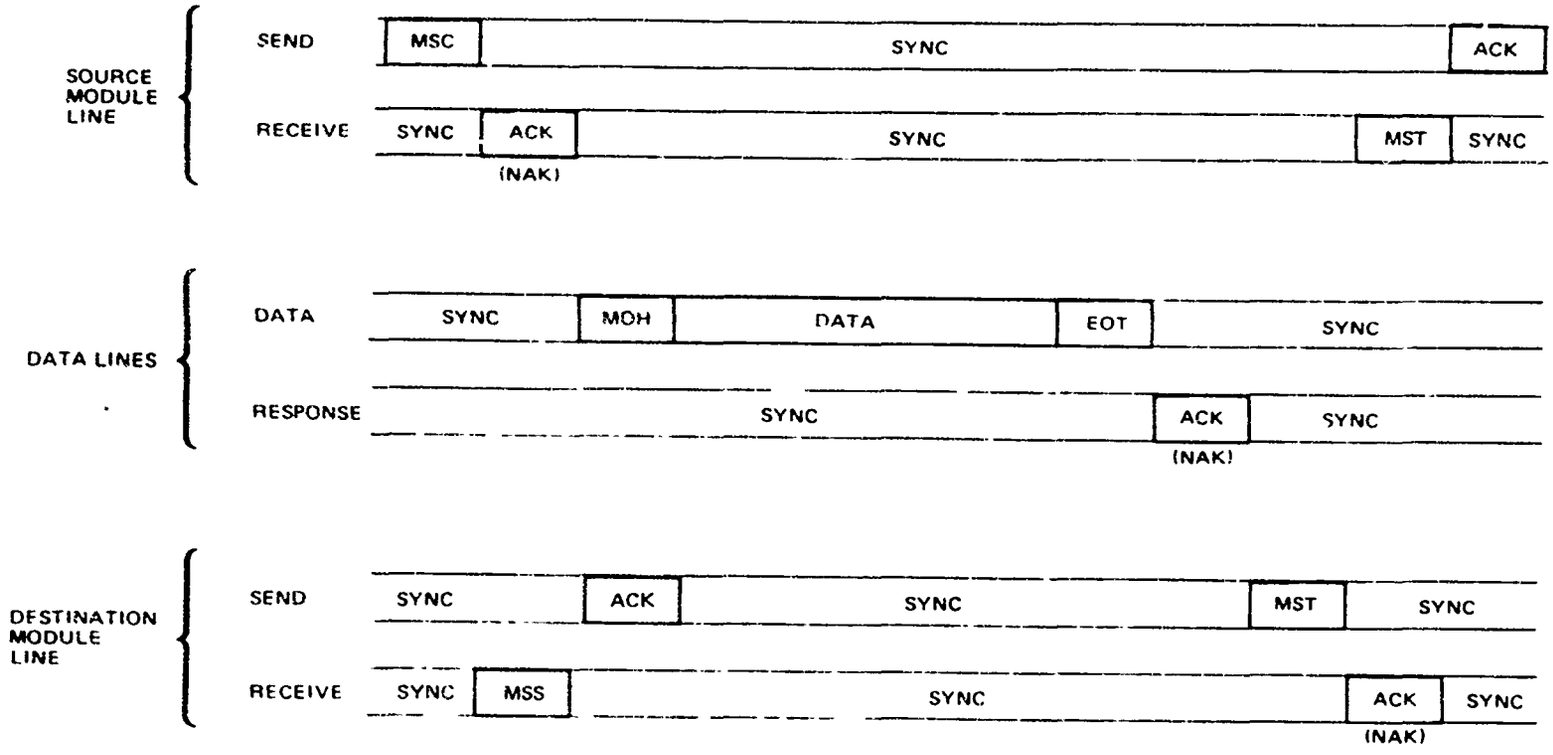
1. Data Lines

   The transmission mode used for the matrix switch system is a full duplex asynchronous operation with automatic error control (module to module level) using one (1) way data transmission. The return direction is used exclusively for module coordination and error control. The lines are physically connected between the system modules and the Matrix Switch and virtually connected via the Matrix Switch.

   Serial to parallel and parallel to serial conversions are performed at each end of the lines by the system module's I/O interfaces. Data is transmitted upon notification by the Matrix Switch Controller that the connection has been made. The data is framed by control information as shown in Figure A.3-6. The module header (MOH field) is made up of several characters which identify the data, the process, and the source module. The EOT field is a single character which indicates the End of Transmission. Based on a 100 characters data block, the module header (2 char) and EOT (1 char) overhead would be about 3 percent of the transmission time. ACK's or NAK's are returned via the return direction line.

2. Control Line

   The control lines are also serial two way lines, however, they go no further than the Matrix Switch Controller. Continuous sync characters are sent to and returned from the Matrix Controller. Refer to Figure A.3-6. When a connect is desired, the requesting module generates a "Matrix Switch Connect" (MSC) request which consists of three characters: a "connect" character followed by two (2) address characters. The receiving module upon receipt of EOT generates a terminate request. ACK's and NAK's are also returned to the requesting module. Adding these control characters to the transmission time adds another 6% of overhead.



- SYNC – CONTINUOUS SYNC CHARACTERS
- MSC – MATRIX SWITCH CONNECT, ADDRESS
- MOH – MODULE HEADER
- EOT – END OF TRANSMISSION
- MST – MATRIX SWITCH TERMINATE
- ACK – ACKNOWLEDGE
- NAK – NO ACKNOWLEDGE - ERROR/NOT AVAILABLE

12677-29

Figure A.3-6. Control Formats and Sequence.

APPENDIX B

ARRAY PROCESSORS

ARRAY PROCESSORS FOR SIGNAL PROCESSING AND SWITCHING APPLICATIONS

In the process of analyzing alternate architectures for signal processing systems and considering the application of hardware types for module implementation, the array processor was seen to provide a "multi-module" function in many signal processing situations Because the array architecture is not as modular as the others considered, it was not included in the basic studies which are the subject of the body of this report. However, its architecture is worth discussion for application to signal processing tasks wherein the total level of system expansion may be accurately forecast (e.g., a unit whose applications could range between a voice encoder and a compatible data modem).

In addition, a parallelism between the array processor's address processing function (to be described) and the message routing functions of switching systems was noted. This similarity is described briefly at the conclusion of this appendix. Further investigation of these application areas is an appropriate topic for related study.

Introduction

Array processors are a very efficient means of implementing digital filters and other functions common to signal processors. This is because the architecture of the array processor (Figure B-1) has been optimized for performing arithmetic (and/or logical) operations on a predeterminable sequence of data points (memory locations).

Conversion of the continous real-time signal input into arrays is the main task of the Control and I/O processor. Further, because the data is to be operated on in a pre-determinable sequence, an address processor is used to generate this sequence and supply the needed arguments to and take the results of calculations from an arithmetic (logic) processor.

The generalized array processor implements the function $\vec{y} = F(\vec{x})$ where $\vec{y}$ and $\vec{x}$ are vectors. The number of elements in $\vec{y}$ is not necessarily equal to the number of elements in $\vec{x}$. The arithmetic processor performs unary or binary operations on the elements or pairs of elements in $\vec{x}$. The address processor handles the indexing implicit in the operation. Since the arithmetic operations use and generate data in a predeterminabie sequence, the address processor can be calculating addresses for the next data point(s) while the arithmetic processor is working on the current one(s). A vector dot product $s = \vec{a}.\vec{b} = \sum_i a_i b_i$ is a special case of the generalized $\vec{y} = F(\vec{x})$. This equation shows up frequently in signal processing and specifically in digital filter expression. It may be noted that to evaluate this expression, the arithmetic processors need only to add and multiply and the address processor does simple indexing. It may also be noted that the array processor can handle multidimensional arrays by treating them as appropriately long vectors and using nested levels of indexing in the address processor.

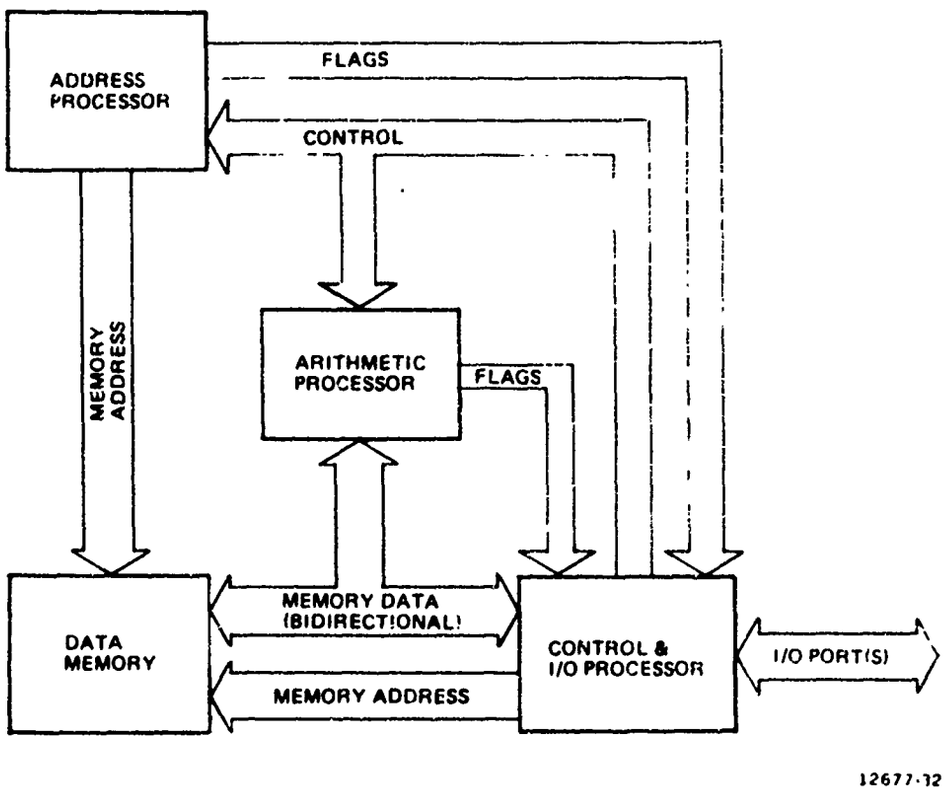


Figure B-1. Generalized Array Processor

Common Problem Areas in Array Processors

In various array processors that were studied, three areas were noted that frequently cause difficulties. These areas are I/O, modularity, and programming. It may be further noted that deficiencies in these areas are not a necessary result of the array processor architecture, but seem to arise from the concentrated effort on the part of the designer to obtain maximum internal operating speed. In some cases this effort has been so concentrated that the other areas have been effectively ignored. In an array processor, performance need not be sacrificed to improve the other factors. The other factors simply need to be given more consideration than has been typical.

Input/Output

Array processors are very efficient at processing data once it is in data memory or a scratchpad, but frequently getting the data in and out of the array processor is slow and/or difficult.

Analog I/O is inherent in a complete signal processor. Typically the input signal is a receiver's i.f. or baseband output, its a.g.c. signal, and the output of other external sensors. The processor's analog output is usually a modulating signal and frequently a variety of control voltages. With this in mind, techniques for implementing analog I/O were investigated. The class of problems wherein a single processing module is the sole or predominant user of the signal in its original form is of special interest for two reasons: (1) Many signal processors fit this class of problems; and (2) Practical conclusions may be made about a feature of the system architecture.

(1) Frequently an analog signal is digitized and applied to a digital filter. The output of the digital filter is then used for further processing. Similarly, a serial bit stream may be generated and applied to a digital filter for waveshaping (bandwidth compression) before conversion to an analog signal.

(2) Analog I/O should be handled directly by the module rather than attaching the analog I/O to a common intermodule communication port. Analog sample rate is often comparable to intermodule communication rate and attaching the analog I/O to a common port unnecessarily absorbs I/O bandwidth on the port.

Implications of this conclusion will be discussed further in a later paragraph. In any event, I/O should be a natural feature built into an array processor, and not something tacked on as a necessary afterthought.

Modularity

Can array processors be modular? At the conceptual level they are. An array processor is a multiprocessor configuration, and each processor can be thought of as a module. Furthermore, a means of modular expansion would seem to be possible. E.G., a simple but nontrivial form of an array processor contains an arithmetic processor, an address processor, and a control and I/O processor. Some array processors use multiple arithmetic units. Some add a second address processor. One address processor then maintains an input queue for the arithmetic processor(s) and the other one maintains the output queue.

The following areas present difficulties with modularization:

Processor/Processor interaction

Multiple address processors tend to be highly interactive.

Mutliple data paths

Full utilization of arithmetic units requires multiple memory modules and the ability to simultaneously route data between multiple pairs of arithmetic units and memory modules. This creates a complex set of switchable data paths and a complex control problem for the address processor(s).

When modules are highly interactive and interconnect is complex, modular expansion at anything other than the conceptual level is very difficult.

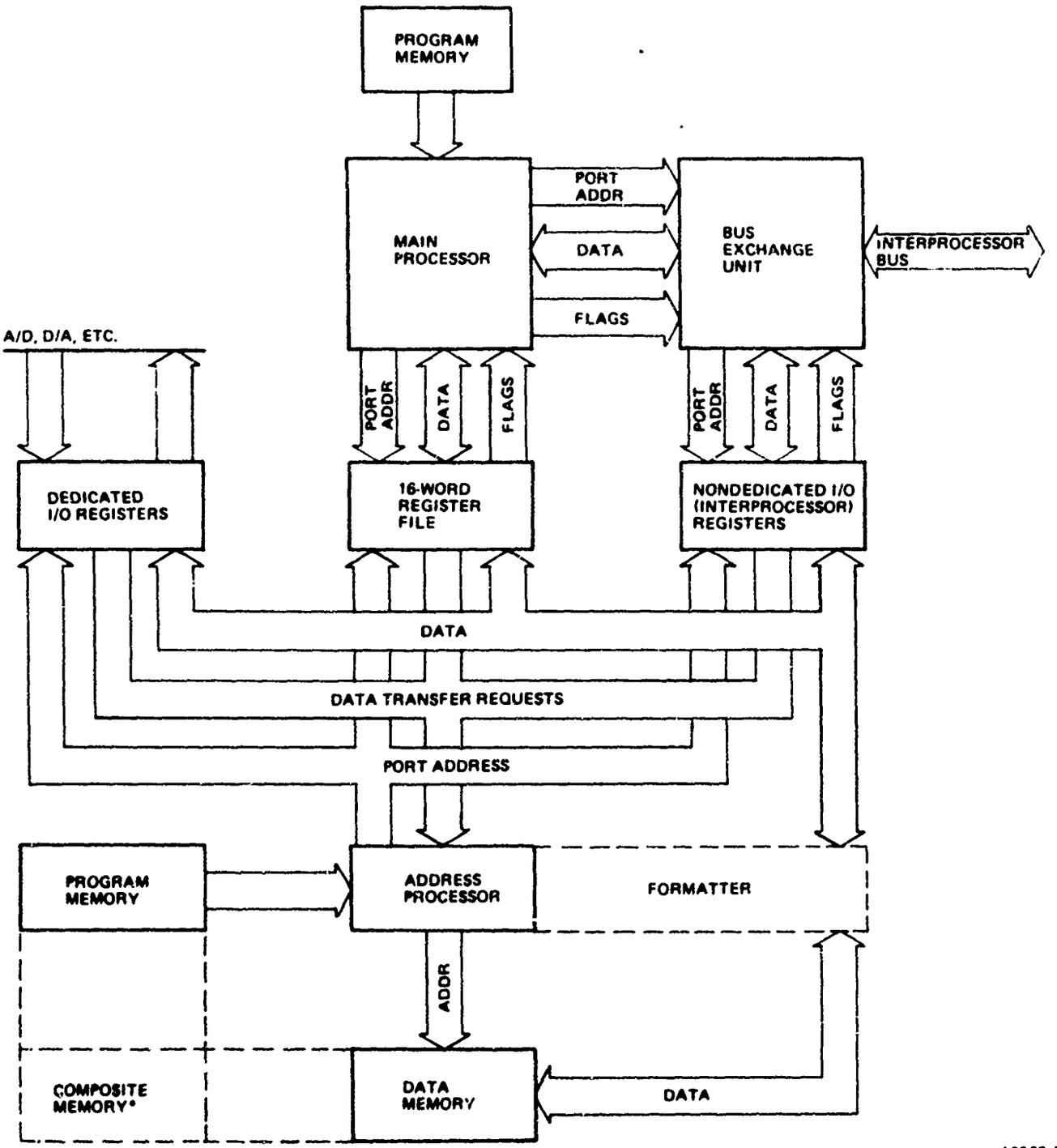
Programming

For reasons related to the above hardware difficulties, most array processors are difficult to program. Highly interactive processors must be concurrently programmed. This is further complicated by the pipelining that is usually done within array processors to increase their speed. It is not unusual, for example, for the programmer to have to explicitly request a memory access, three machine cycles prior to an instruction that uses data read from memory. The pipelining of memory accesses can be made transparent to the programmer if a means exists for defining the arrays and the associated addressing algorithms prior to execution time. If the address processor is made to be data driven, then pairs of arithmetic and address processors need not be programmed concurrently. The address processor is simply a slave to data requests from the arithmetic processor.

A Solution

Figure B-2 is a block diagram of an array processor configuration which includes a natural means of doing I/O, which has two levels of hardware modularity, which promotes modular software, and which is easy to program. In this configuration, arithmetic and control functions are performed by the main processor. Data Fetch and I/O functions are performed by the address processor. This provides a very simple and effective means for the array processor to do I/O. I/O devices simply make data requests to the address processor. The address processor then services these requests just as it would a data request from the arithmetic processor. Multiple array processors may be connected together via the interprocessor bus. Communication between array processors occurs whenever a main processor specifies an access of an array which resides in the data memory of a different array processor. The address processor services data requests from the bus exchange unit in the same manner as any other data request. Note that the I/O mechanisms in this configuration meet the requirements given in the earlier section on I/O. When multiple array processors are used in a system, the dedicated I/O ports are used to bring the I/O directly into the array processor that is the sole or primary user of the I/O. The I/O is not placed on the interprocessor bus.

The top level of modularity for this configuration is that is provides a means for easily connecting multiple array processors together via the interprocessor bus to form a larger system. In fact, other types of processors could also be connected to the interprocessor bus.



*COMPOSITE MEMORY IS INCLUDED ONLY IF THERE IS A REQUIREMENT TO BE ABLE TO WRITE INTO PROGRAM MEMORY

**A FORMATTER IS INCLUDED ONLY IF THERE IS A REQUIREMENT FOR FORMAT CONVERSION SUCH AS BETWEEN FIXED POINT AND FLOATING POINT FORMATS.

Figure B-2.

This configuration also has a second level of modularity in that it allows multiple address processors to be used to increase address processing speed. The address processors do not interact with each other except that they contend for common data memory. The address processors, therefore, do not have to be programmed concurrently. In fact, if the address processors are identical and each contains all of the necessary address algorithms and array definitions, then the very existence of multiple address processors is transparent to the programmer. Whenever a data request is made, the first available address processor will acknowledge and process the request. To the rest of the system, the multiple address processors look like a single high speed address processor.

Multiple address processors become desirable in two situations: (1) The array processor requires a large amount of I/O; (2) A very fast main processor is used. Using multiple address processors in the first case is a simple extension of the requirement stated earlier that I/O must not overload a common facility. In both cases, the capability for modular expansion at the address processor level makes it easier to achieve optimum cost vs. performance.

There are cases where nonidentical address processors may be desirable. Dedicating an address processor to the servicing of particular ports makes a slight reduction in necessary hardware. Address processor program memory requirements may be reduced if each address processor services only a subset of the total number of defined arrays. The implications that nonidentical address processors have on the software is discussed in a later paragraph.

The following program for the array processor shows how array definition is accomplished and explicitly performs the dot product of two vectors. Note that the pipelining of the data fetch is transparent to the programmer, and that the main processor and address processor are not programmed concurrently. In fact, a certain amount of modularity is inherent in this program. Not only are the address processor and main processor coded independently, but the different address algorithms to be executed by the address processor are also coded independently.

A brief description of some of the instructions follows the sample program.

```
        DEFINE ADDRESS ALGORITHM 1
            instruction statements
              .
              .
              .
        DEFINE ADDRESS ALGORITHM N
              .
              .
              .
        DEFINE ARRAY 1
            parameter statements
              .
              .
              .
        DEFINE ARRAY N
              .
              .
              .
START:  READ    ARRAY1 VIA PORT1
        READ    ARRAY2 VIA PORT2
        WRITE   ARRAY3 VIA PORT3

  DOT:  CLR     R1

 LOOP:  MOVE    PORT1,A
        MULT    PORT2,A
        ADD     A,R1
        TEST    ARRAYEND
        IF FALSE GO TO LOOP
        MOVE    R1,PORT3
              .
              .
              .
            other processing
              .
              .
              .
        WAIT FOR FRAME TIME
        GO TO DOT
```

ADDRESS PROCESSOR FUNCTIONS

MAIN PROCESSOR CONTROL FUNCTIONS

MAIN PROCESSOR ARITHMETIC FUNCTIONS

Instruction Description:

DEFINE ADDRESS ALGORITHM

Instruction statements following this definition statement generate code for the Address Processor's program memory. The address processor executes these instructions whenever the arithmetic processor accesses an array which uses this algorithm to determine the addresses of elements in the array.

DEFINE ARRAY

Parameter statements following this definition statement generate entries in an array definition block to reside in the address processor's program memory. The array definition block contains the following information.

Address (in data memory) of the first word of the first element of the array.

Access control word. (May point to a list of users authorized to read/write.)

Address (in address processor's program memory) of the address generation algorithm.

Algorithm dependent parameters — number of words per element, element spacing, physical length of circular buffer, etc.

READ

Moves a word from the main processor to PORT 0 of the 16 word register file. This word contains the number of the array definition block and the data port number. Upon receipt of this control word. the address processor initiates the appropriate address algorithm and moves the first word of the array from data memory to the specified data port.

WRITE

Same as READ but no data is moved

TEST ARRAYEND

Test a completion flag associated with the register file which is maintained by the address processor.

IMPACT OF MULTIPLE ADDRESS PROCESSORS ON SOFTWARE

As mentioned earlier, the existence of multiple identical address processors has no impact on the software, but nonidentical address processors do. If, for example, a unique address processor is dedicated to I/O and/or interprocessor communications and another one is used to service the main processor. the impact on the software is minimal. The only requirement is that a means must exist for directing the DEFINE ADDRESS ALGORITHM and DEFINE ARRAY statements to generate code for a particular address processor. If, however, an address processor is dedicated to particular main processor ports. the programmer must be constantly aware of which ports are being used in order to insure parallel processing. If the address processors are further specialized so that not all array definitions and/or algorithms are present in each processor, there is a significant additional burden placed on the programmer.

As was mentioned earlier, multiple address processors become desirable in two situations: (1) the array processor requires a large amount of I/O; (2) a very fast main processor is used. The second case is of interest since there are ways other than brute force to build a fast main processor. Let us reconsider the main program loop in the sample program.

```
LOOP:   MOVE    PORT1,A
        MULT    PORT2,A
        ADD     A,R1
        TEST    ARRAYEND
        IF FALSE GO TO LOOP
```

A main processor that executed these instructions could generally be supported by a single address processor that had a good instruction set and a cycle time equal to that of the main processor. If, however, a three address machine were used for the main processor; the main program loop could be written as follows:

```
LOOP:   A=PORT1*PORT2
        R1=R1+A
        IF (ARRAYEND=FALSE) GO TO LOOP
```

The three address machine accomplishes the task in three machine cycles instead of five and would require two "ordinary" address processors to support it. Aside from its improved speed, the use of a three address machine is desirable for the main processor for its ease of programming. As can be seen from the sample programs, the three address processor uses a higher level machine code. The three address machine is particularly feasible in this array processor configuration since its addresses are port and internal register addresses rather than memory addresses. Since fewer bits are required for each address specification, the instruction word can actually be shorter in length than a traditional two address machine that makes direct memory accesses.

ARRAY PROCESSORS AND SWITCHING SYSTEMS

Array processors were investigated because of their direct applicability to signal processing. As the modular architecture study progressed, however, it became apparent that array processing techniques could be applied to switching systems too. The address processor can, for example, do dynamic memory allocation and can service the traditional "mailboxes" in a manner that is transparent to the rest of the system. Various table lookup and search algorithms could also be implemented. The algorithms would be easy to access from the arithmetic processor but further investigation would be necessary to develop programming techniques that are natural to use and that would result in parallel processing in the address and arithmetic processors. Without such techniques, the arithmetic processor might end up idling while it waits for the address processor to find a table entry. This may be able to be justified on the basis of ease of programming but it certainly doesn't utilize the full potential of the array processor. Further study in this area is indicated. Further study is also needed in the area of definition of instruction sets for the address processor and arithmetic processor. For switching systems, the high speed multiply and other such extended arithmetic capabilities that are highly desirable for signal processing might well be sacrificed in favor of powerful logical, bit manipulation, and decision making instructions. In the address processor, capabilities for multilevel indirect addressing with both preindexing and postindexing are probably desirable. As was stated before, further study is indicated.

APPENDIX C

PARAMETERS FOR MODULE SPECIFICATION

PARAMETERS FOR MODULE DEFINITION

C.1 GENERAL DESCRIPTION

The specification for a module is an iterative function. The first step is definition of the algorithm and definition of the preliminary design specifying as many parameters as possible. As the modeling of the equipment progresses and as the design progresses, more of the parameters become apparent and can be specified. Eventually the parameters in the module definition will be used to form the body of a module specification used to completely define a module physically, electrically and functionally. Some cases exist when a module is completely specified initially. This case exists when more of the external operating environment for the module is known. In this case, the problem is only one of designing a module to meet the specification. The following sections describe which module parameters need to be specified.

C.2 MODULE DESCRIPTION

Description of the function performed, i.e., Output = Input x Transfer Function. Input/Output oriented block diagram identifying data, control, and timing requirements. A discussion related to this diagram should indicate the system designer's options, e.g., where certain control signals are always required; where they are not. This discussion will also stipulate external requirements, e.g., on the system executive; it should also describe the implications of software implementation. It should specify the type of control and the number and type of interrupts, if required. For software, the control entry requirements must be specified. Data formats and number systems used should be specified as the design progresses.

C.3 MODULE INPUTS

(a) Format
(b) Input Port, master/slave, access time
(c) Multiplex requirements, e.g., is more than one signal input via the same port?
(d) Timing (rate), timing will frequently vary with application. Specify input rate in normalized terms, e.g., 10 input words are required per output.
(e) Timing (sequence), relationship of data and control inputs. (Design Note: Description should specify speed independency whenever possible).
(f) Addressing, identify all vector addresses used for control, all data input addresses and engineering inputs which can be to specify relative addressing within the module.
(g) Other inputs, if non-address inputs are used, specify their reason for occurrence, the frequency, format, etc.

C.4 MODULE OUTPUTS

(a) Format
(b) Output Port, master/slave, access time
(c) Multiplex control requirements when multiple outputs share a single port.
(d) Timing (rate), normalized (see Inputs)
(e) Timing (sequence), dependencies (see Inputs)
(f) Addressing, identify all vector addresses used for control of other modules, all data output address used in either master or slave states.
(g) Other outputs, if non-addressed outputs are used, stipulate their reason for occurrence, their frequency, format, etc.

C.5 TRANSFER CHARACTERISTICS

(a) Throughput delay. This will typically indicate how many input words and commands are required to produce an output. (Processing Time)
(b) Scaling. If not clearly defined in C.2 and C.3, data dependencies should be explicitly defined here.
(c) Fault detection. Consider various input failure modes (see below) and define output indication.

Input Failures	Indication
(1) Power	(Meter, Lamps, & other Indicators)
(2) Facility signals	
(3) Out of sequence controls	

C.6 OTHER REQUIREMENTS

This section specifies all power supply, cooling or other external requirements the module has. This section is expanded as the module design progresses.

APPENDIX D

ARCHITECTURE SELECTION CRITERIA

ARCHITECTURE SELECTION CRITERIA

Whenever the importance of individual line items in the selection criteria may be dependant on the area of application, the selection criteria were independantly applied to two categories of applications — switching systems (SW) and signal processing systems (SP). Each of these categories is broad enough to be of general interest but narrow enough to allow reasonably precise comparisons. Those line items which are considered to be particularly important to each of the categories are indicated by two columns of asterisks at the left hand edge of the table of selection criteria.

I. COST. The cost to be considered is true life cycle cost including costs of acquisition, operation, and logistic support.

SW	SP	
		1. Required hardware/software (overhead)
*	*	(A) What is needed to support minimum configuration?
*		(B) What is needed to support each additional module?
		2. Flexibility If a system design is flexible enough to be used for other systems, then development costs can be amortized over more than one contract. Production costs for the first contract may be handled in a similar manner if the increased flexibility results in a reduction of production costs when mulitple contracts are considered.
		(A) How well are widely varying performance requirements handled?
*	*	(B) Is it easy to go from the general to the specific? (This is related to ease of design.)
*	*	(C) Is the specific still general relative to the class of problems?
		3. Adaptable within a configuration If one assumes that it will be desirable or necessary to improve (modify operation or add features) a given system during its expected lifetime, then the adaptability of the system has a significant impact on life cycle cost of the system. Furthermore, a system which may be easily improved will probably have a longer life expectancy.
*		(A) How is growth in hardware or software achieved? (New design or repetition?)
	*	(B) How can field changes be tested and implemented?
*	*	(C) What limits does the design place on upper signaling rates, response times, dynamic range, memory or device addressability, terminal I/O, etc.? (Allowance for two to one growth over the lifetime of signal processing equipment is probably adequate.)

SW	SP	
		3. Adaptable within a configuration (Continued)
	*	(D) Can the design accommodate technology change in signaling techniques, filtering techniques, A/D dynamic range, etc.? (What is the probability of such change?)
		4. Ease of design
*	*	(A) What hardware/software exists that is similar or identical to the required modules?
		(B) Can "standard" components/languages be used?
		(C) Are both the hardware and software natural to use or does one have to continually be "clever"?
		(D) Can it be tested simply?
*	*	(1) Can it be tested incrementally?
*	*	(2) Can it be easily prototyped and/or simulated?
		5. Ease of documentation
*	*	(A) Is a common interface used for software and/or hardware modules?
	*	(B) To what degree is the hardware/software self documenting?
*	*	(C) Did the original design light the way for future changes? This is necessary because personnel will usually be different between an initial design and the redesign event.
		6. Manufacturability
*	*	(A) Is it state of the art? (Consideration must be given to the time of application, the necessity of the technique, and must include an effort to forecast industry trends.)
*		(B) Does the depth of the logic require long test sequences?
*		(C) Does "cleverness" of the design require high level test technicians?
*	*	(D) Is it machine diagnosable? (Machine diagnosis may reduce but not eliminate the impact of the above two items.)
*	*	(E) Can design phase tests be used to partially or wholly satisfy production test requirements?

SW	SP	
		7. Maintainability
	*	(A) Can fault isolation be easily achieved? (Busses make this difficult; also relative addressing of any kind, virtual machines, and any kind of multiplexing is less than optimum.)
*	*	(B) Can test facilities be built in naturally?
	*	(C) Cabling, connections, etc. Do these force unnatural mechanical designs which are hard to maintain?
		(D) Can the mechanical design be transferred easily? (This is a measure of the simplicity of connection.)

II. RELIABILITY

		1. Ascertain MTBF. (Relates to maintenance costs also.)
*	*	2. Ascertain the number of single point failure locations. (Probably much more important than basic MTBF.)
*	*	3. Ascertain types of failure modes. Failure modes must identified in terms of their system impact (percent loss of function). Critical sections must be analyzed for failure detectability.
		4. Is the design pushed to technology limits?
		(A) Timing constraints
		(B) Environmental sensitivity
		5. What is the maturity of the technology itself?

III. PERFORMANCE

*		1. Data processing capability (throughput and latency)
		2. Control requirements (forced overhead)
*		(A) Data interchange and buffering.
*		(B) Required protocol.
*	*	(C) Data routing requirements. (Does data from 'a' to 'b' have to go via 'c'?)

SW	SP	
		IV. CUSTOMER REQUIREMENTS
	*	1. Power, cooling, cabling
*	*	2. Size, weight, form factor
*		3. Flexibility, adaptability (see sections I.2 and I.3)
		4. Security
*	*	(A) Red/Black isolation
		(B) Security to single point failure
		(C) EMI/EMC
		(D) TEMPEST

APPENDIX E

SIGNAL PROCESSING TARGET SYSTEM

DESCRIPTION OF SIGNAL PROCESSING TARGET SYSTEM

The general class of signal processing equipment modeled in this study is confined to communication links which transfer data or digitized voice. With this type of equipment both signal processing and communication processing may co-exist. We have restricted our analysis to the receive terminal end of the system. Information which is input to this equipment is carried on a receiver IF frequency with the information bandwidth limited to approximately 3 KHZ. This information is demodulated using complex arithmetic techniques and processed further according to the data type. Parameters which may vary within this type of equipment are: Information rate, modulation index, encrypted data or bandspreading, coding techniques, error detection and correction processing, and message formatting.

The equipment modeled in this study receives an "IF" frequency of 7.5 KHZ which is modulated by a phase continuous FSK signal with a modulation index of 0.5. The modulation rate varies from 100 Hz to 1600 Hz. Correlation or integration periods of 1/2 second to 2 seconds are included in the module library. The two coding types considered were antipodal (Complementary Orthogonal CO) and M-Ary where M=65. Characters are grouped using the detected symbol stream such that two characters are received every fifteen correlation intervals. Appendix I (and Figure I-1) describes a partitioning of this type of equipment which is not necessarily optimum but provides a basis for analysis of the architecture using the simulation model.

APPENDIX F

SWITCHING TARGET SYSTEM

MESSAGE SWITCH APPLICATION

I. General

The following paragraphs describe, briefly, a 12-line message switch providing austere store-and-forward service of both record and data traffic to a variety of connected terminals and between other switches. This switch is similar to the TRI-TAC Unit Level Message Switch (ULMS).

II. Interface Definition

The switch interfaces with 12 subscribers, a Traffic Service Position (TSP), a Tactical Communications Control Facilty (TCCF), and another ULMS. Each of the 12 subscriber interfaces can handle either asynchronous NRZ or synchronous conditioned diphase signals. For the purpose of this application, an arbitrary mixture of these signal types are used as shown in Figure F-1. A mixture of I/O message rates suggested for a test bed configuration in the ULMS specification are used for this application. A description of the signals for each of the interfaces are given in the following paragraphs.

A. Subscriber Interfaces (12)

1. Asynchronous Digital Interface
   a. Balanced NRZ
   b. Rates: 110, 150, and 300 Baud
   c. Received Signal
      Level: 0.05 to 3.0 Vp-p
      S/N Ratio: 20 dB in signal bandwidth (white noise)
      Timing: Bit-by-bit asynchronous, start/stop codes
   d. Transmitted signal
      Level: ±0.8 to ± 6.0 Vp-p
      S/N Ratio: 80 dB in signal bandwidth
      Timing: Bit-by-bit asynchronous, start/stop codes
2. Synchronous Digital Interface
   a. Conditioned diphase
   b. Rates:

Modulated Rate (bps)	Information Transfer Rate (bps)
600	600
1200	1200
2400	2400
16000	*8000, 16000
32000	*16000

*When MR/ITR ratio is 2/1 (16/8 or 32/16) dual bits (00 or 11) shall be processed or generated.

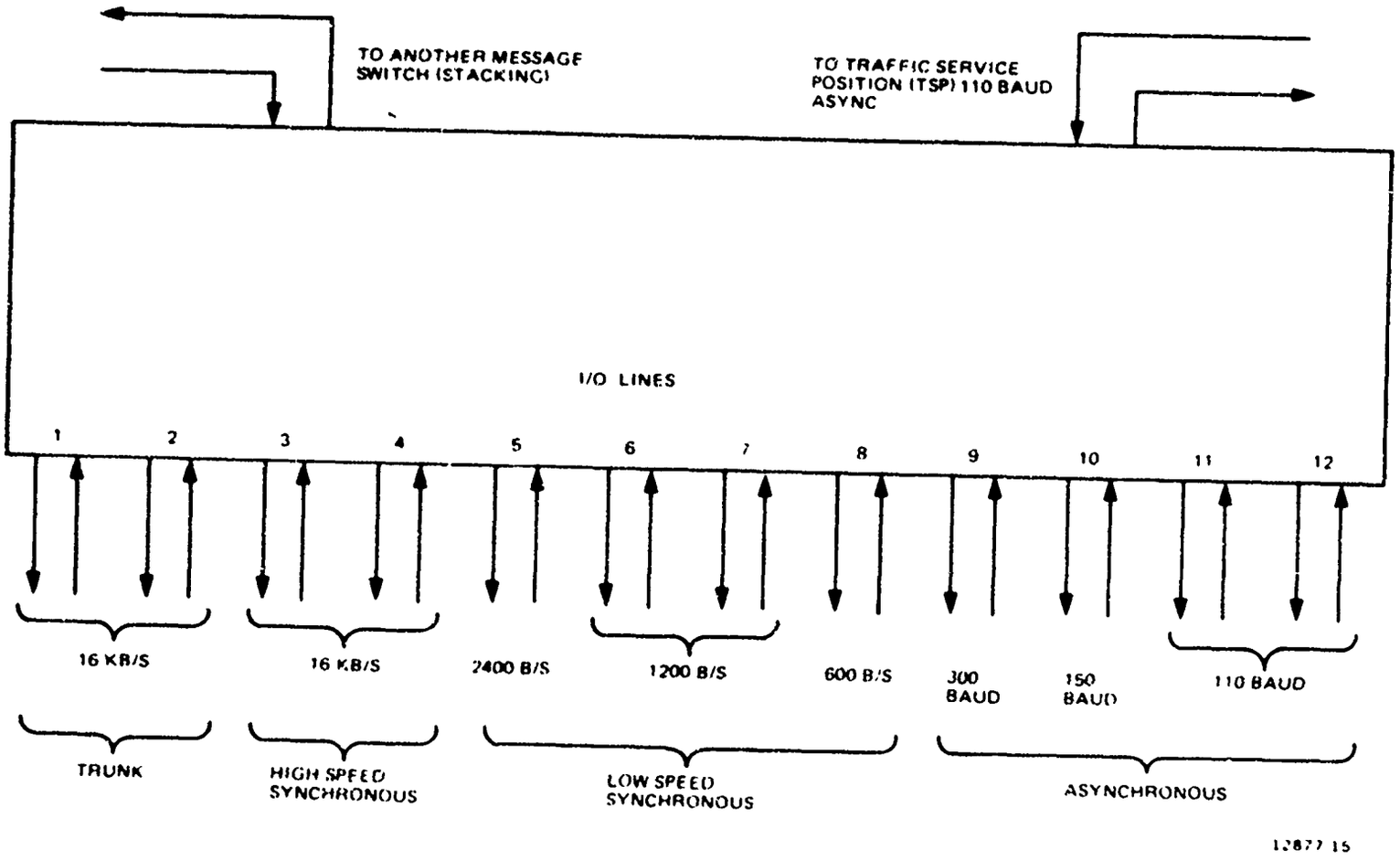


Figure F-1. Message Switch Target System.

c. Received signal

Level: 3Vp-p or less (capable of sensing transmit signal over 3200 meters of WF-16 cable)

S/N Ratio: 20dB in signal bandwidth

d. Transmitted signal

Level: 3±0.3 Vp-p terminated in 135 ohm

S/N Ratio: 80 dB in signal bandwidth

B. Traffic Service Position (TSP) Interface

1. Dedicated Termination
2. Asynchronous Digital Signals, NRZ
3. 110 Baud ASCII
4. Mode II - full duplex, no error or channel controls
5. Signal levels as described under Asynchronous Digital Interface paragraph above.

C. Tactical Communications Control Facility (TCCF) Interface

1 Dedicated Termination
2. Conditioned diphase signals
3. Rate: 2000 bits/sec.

D. Another ULMS Interface

1. Direct (hardware) interconnection
2. Dedicated termination

III. Message Processing

A. Message Store and Forward

The message switch shall receive incoming messages and subsequently shall transmit these messages directly to subscriber terminals or to other message switches for ultimate transmission to destination terminals. Traffic awaiting transmission shall be stored as intransit traffic.

B. Message Security

The message switch shall handle the following classification levels:

TT - TOP SECRET
SS - SECRET
CC - CONFIDENTIAL
UU - UNCLASSIFIED

Each incoming and outgoing channel shall be class marked for a maximum security level. If a non-perishable message is received that exceeds the security class mark of the channel, the message is delivered to the traffic service position for disposition. If the message is perishable, it is discarded.

C. Precedence Feature

The Message switch shall recognize five precedence levels, as follows:

Y - ECP
Z - FLASH
O - IMMEDIATE
P - PRIORITY
R - ROUTINE

Messages shall be queued for transmission first by precedence level, then by order of arrival. Service messages shall be placed in queue ahead of all subscriber-originated messages of equal or lower precedence. A message in process of transmission shall not be interrupted for transmission of another message of any higher precedence level.

D. Throttling

Throttling is the implementation of certain procedures to regulate peak load.

Throttling shall be implemented automatically upon reaching preset thresholds for percentages of in-transit storage capacity utilized. There are three threshold levels, each of which are adjustable by the supervisor. Passage of these levels shall result in the rejection successively, of incoming messages of ROUTINE, PRIORITY, and IMMEDIATE within precedence, order of rejection shall be: first, messages originated from local subscriber terminals, and second, messages incoming from trunks. FLASH and ECP messages shall never be rejected during any throttling condition.

E. Processing Capacity

The 12-line message switch shall have sufficient processing capacity and message buffering to handle concurrently total input and output information as shown below:

	1 second Interval (8-bit characters)	1 hour Interval (8-bit characters)
Input (Receive)	3×10^3	2.1×10^6
Output (Send)	3×10^3	9.84×10^6

F. Processing Time

The message switch shall have a mean processing time of no more than 0.10 seconds for all messages processed. For ECP and FLASH messages, no more than one in 10^3 shall have a processing time greater than 0.50 second. Processing time is defined as the time from reception of the last bit of EOM for a message until that message is placed in in-transit storage plus the time from the instant of availability of a transmission channel for the message until the first bit of the SOM, tactical header, or path field for the message is output. The time which a message remains in in-transit storage shall not be included in the processing time.

IV. Message Description

A. Codes

The message switch shall process messages on a character-oriented basis. Characters shall consist of 7 bits of information and an 8th bit to achieve odd parity. Line and message protocol shall utilize USASCII characters with even and odd parity, respectively. Message text will be transparent.

B. Message Protocol

Message protocol elements utilized by subscribers shall be a start-of-message indicator, a tactical header, and an end-of-message indicator.

1. Start-of-Message Indicator (SOM)
   a. Asynchronous loops
      Four "W" characters, two carriage return characters, and two line-feed characters
   b. Synchronous loops
      SOM is inherent in the line protocol employed between the subscriber device and the switch (See SECTION VIII).
2. Tactical Header
   a. Precedence (1st character)
   b. Classification (Security) (2nd and 3rd characters)
   c. Message Type (4th character)
      P - Perishable
      N - Nonperishable
   d. Message Text Code and Format (5th character)
      A. ASCII; teletypewriter
      B. Binary text; standard EDPE system parity check for entire message text
      C. Unspecified; fixed record length, 80 characters per record, series format
      D. Unspecified; variable record length, up to 1200 alpha-numeric characters per record.
      E. Binary text; odd EDPE system parity check for text characters only
   e. RESERVED - NULL (6th - 8th characters)
   f. Originating subscriber terminal tactical routing indicator (9th - 11th characters)
      3 allowable ALPHA characters (Q disallowed)
   g. Start-of-routing indicator (12th character)
      HYPHEN (-)
   *h. Destination addressee terminal tactical routing indicator (13th character - as required)
      Three allowable ALPHA characters each, 45 characters max (Q disallowed).
   i. End-of-rounting indicator (1 character, position as required)
      PERIOD (.)

*Maximum number of destination addresses allowed shall be 15. The length of the tactical header will vary from 16 to 58 characters, depending on the number of destination addresses

3. End-of-Message Indicator (EOM)
   The EOM shall be QQQXXX where XXX is the tactical routing indictor of the originating subscriber terminal.

V. Message Validation

All subscriber originated messages shall be validated by the message switch. Validation checks shall be as follows:

A. SOM Indicator
B. Precedence
C. Classification
D. Message Type
E. Message Text Code and Format
F. Originating Subscriber Terminal Tactical Routing Indicator
G. Start-of-Routing Indicator
H. Destination Addressee Terminal Tactical Routing Indicator(s)
I. End-of-Routing Indicator
J. Message Text
   (Parity-odd; max. chars. -1200)
K. End-of-Message Indicator

VI. Logging

The message switch shall maintain a log for nonperishable messages. It shall be logically and electrically independent of intransit storage. Information contained in the log shall be as follows:

A. Path Field
B. Tactical Header
C. Time of Receipt
D. Time and manner of disposition for each message copy.

VII. Maximum Message Traffic

A. 20000 perishable messages handled on Line 1 during busy hour
   1. The messages arrive at regular intervals, one every 180 milliseconds
   2. Each message is one 80-character block long

B. 2500 non-perishable messages are sent on all other lines during busy hour
   1. Messages arrive at random intervals.
   2. These messages range in length from 1 to 16 blocks long, with a mean of 4 blocks (320 characters).

VIII. Message Layout

The basic message layout (Figure F-2) reflects the message as it will appear in main memory (ASCII). A line handler (see Figure J-1) is responsible for reformatting each message it receives in this format in main memory. The location for the message in main memory is acquired by the line handler from its "mailbox". The line handler will probably begin transfering the message to main memory after it has processed the header and before it has received all of the message text. This implies that the line handler

must count the characters in the message and set the message length field after the message transfer is completed. The line handler will also place a date/time stamp in the message prefix. The date field will consist of the last digit of the current year followed by the three digit Julian day. The arrival time field will occupy four character positions and contain the time of day in hundredths of a second expressed in binary.

IX. Functional Modules

The message switch shall contain the following functional modules:

A. Line Interface Modules
B. Input Service Modules
C. Output Service Modules
D. Code Translation Module
E. Message Storage Module
F. Message Logging Module
G. Validation Module
H. Routing Module
I. System Control Module
J. Memory Management Module
K. Error/Recovery/Alarms Module
L. Initialization Module
M. Diagnostics/Test Module

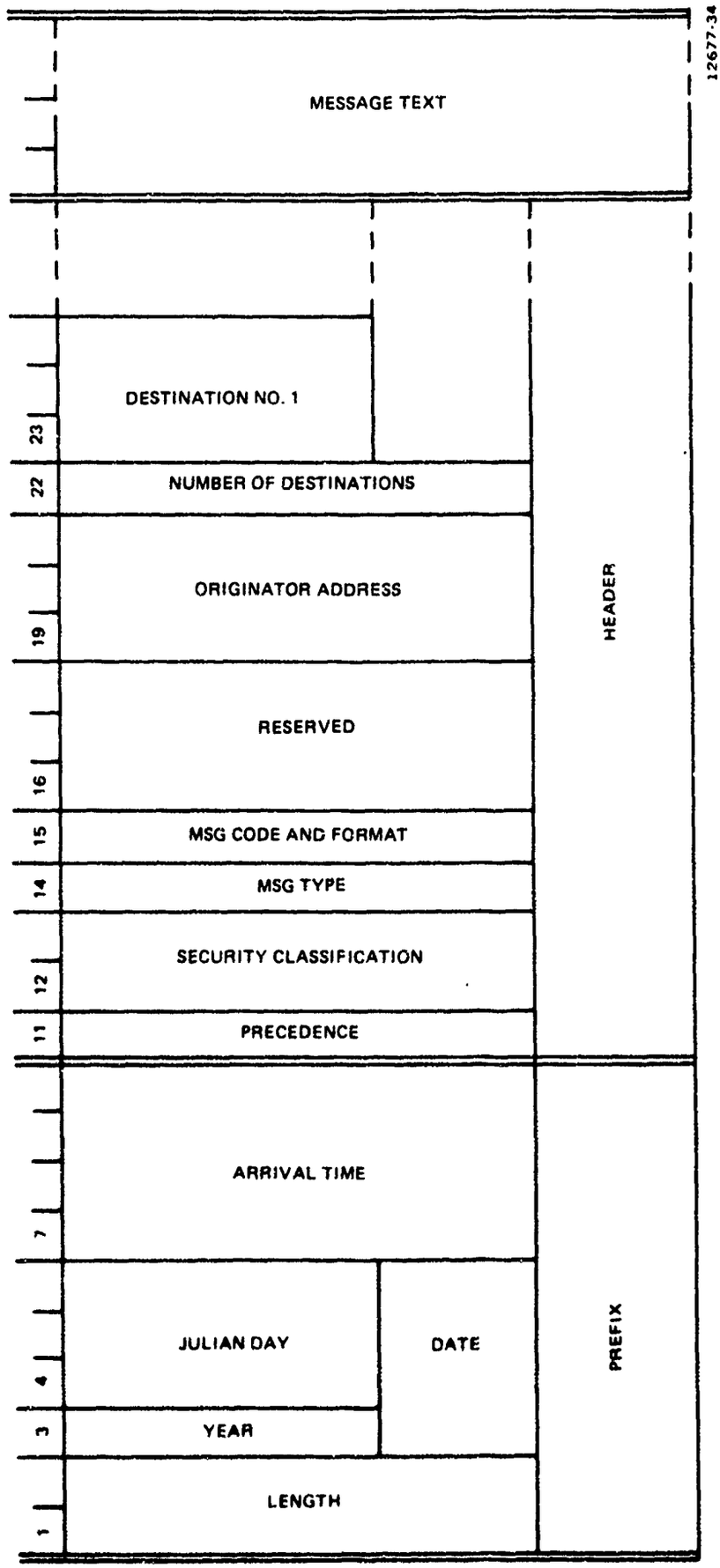


Figure F-2. Basic Message Layout.

APPENDIX G

ARCHITECTURE SELECTION FOR SIGNAL PROCESSING SYSTEMS

I. Summary

The architecture used for the balance of the studies of modular signal processor design is the parallel bus system described in Appendix A.1. The selection criteria (see Appendix D) were applied with the aid of a candidate signal processing target system (Appendix F) by separate judges in the areas of cost, performance, and reliability. The results of these comparisons are shown graphically in Figure G-1 and G-2. In the area of cost, the importance of a particular line item is determined by its contribution to total cost. In the area of reliability, the judge also indicated which item was most important and which was least important. In the area of performance, all items were judged to be equally important. Customer requirements were not included in the analysis due to their total dependence on application.

Figure G-1 shows the percentage of the total score within each category attributed to each architecture. This figure is based on a best, medium, worst scoring for each line item in the selection criteria. It also assumes that best, medium, and worst are geometrically related (e.g., best =4, medium =2, least =1) characteristics. For raw scores see Table G-1. Figure G-2 serves to interpret the raw scores by indicating the "best/worst" distribution among architectures; perhaps thereby providing a measure of system design risk.

II. Architecture Comparison by Cost

When a specific application is known, dollar figures can be calculated or estimated for each line item in this section. The importance of each line item is then precisely determined by its contribution to total cost. In order to keep this comparison independent of application, costs of required hardware are given in terms of the number of circuit cards needed when standard currently available components are used. Costs of software are given in words of microcode, lines of assembly language, etc. The level of coding assumed is that which is commonly used today. This allows accurate estimates to be made now and allows for extrapolation of these estimates to account for coding at whatever level of language is desirable; e.g., HOL. To keep the comparison independent of system size, costs are given on a per module basis where applicable. Where further application dependency exists, relative trends and tendencies are given. Wherever possible, linearity or degree of non-linearity and the existence and magnitude of discontinuities is indicated.

Frequently within this section, the acronyms BEU, MIC, and MIU are used. These are the interface hardware for the parallel bus, matrix, and serial bus respectively.

1. Required Hardware/Software (Overhead)
   - A. What is needed to support minimum configuration? (Includes interface for first two hardware modules)

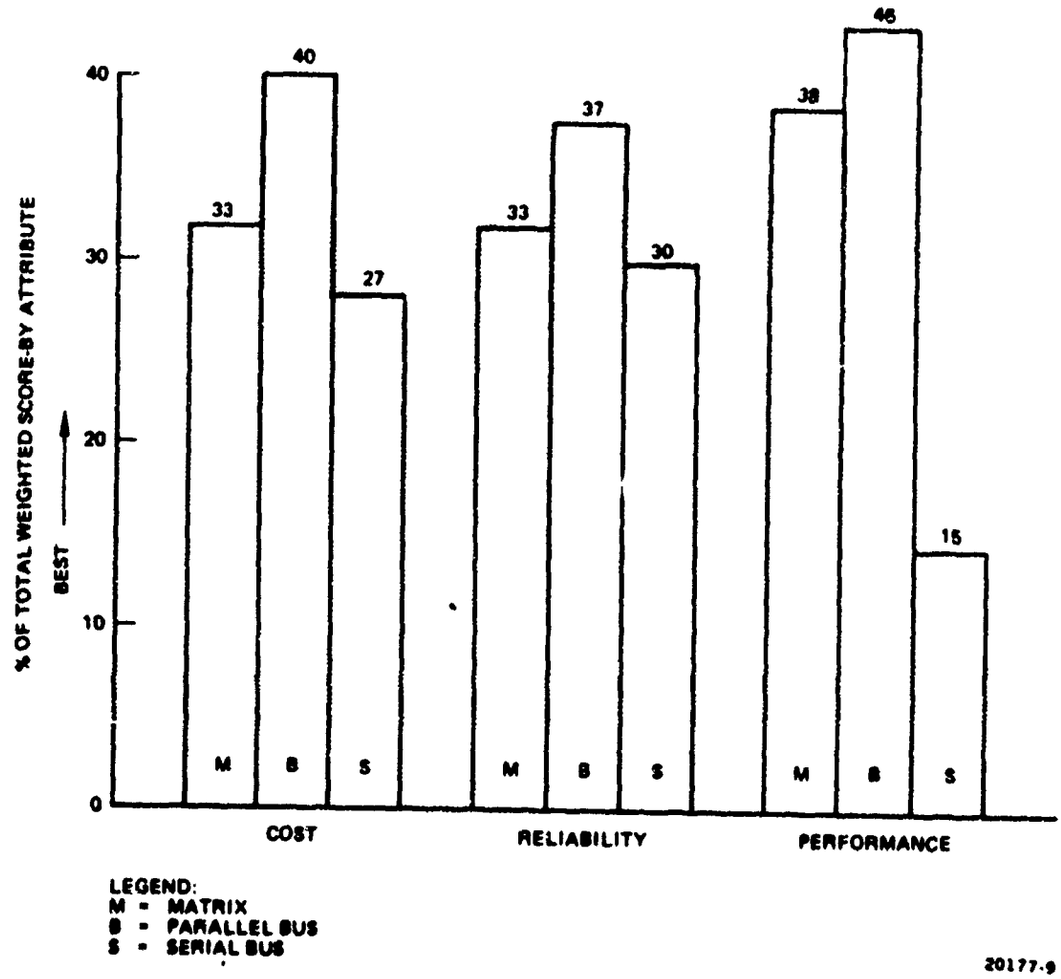


Figure G-1. Architecture Comparison Summary — Totals.

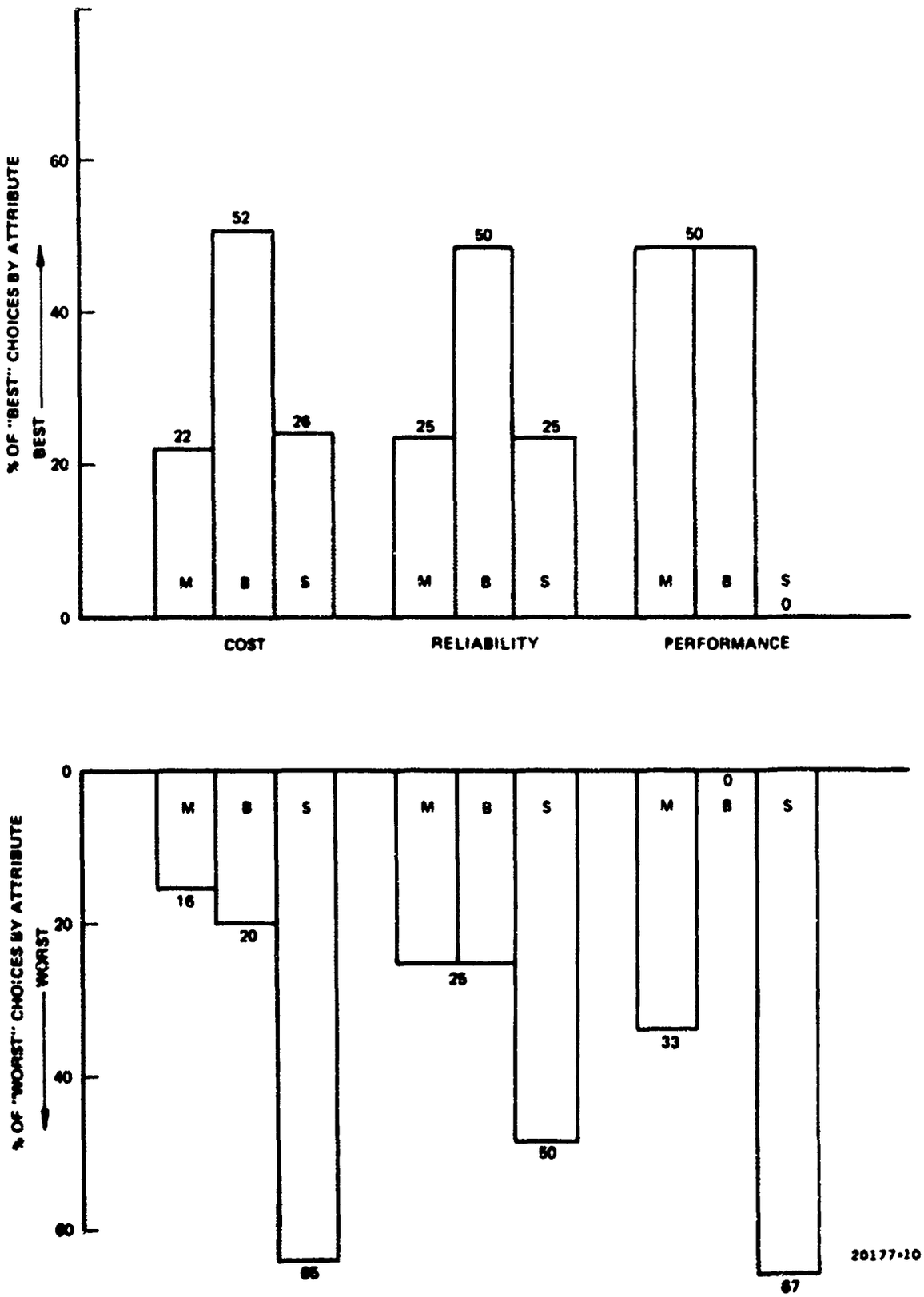


Figure G-2. Best Worst Comparison by Attribute.

Table G-1.

			CHOICE			WEIGHTED SCORES		
			BEST	MID.	WORST	MATRIX	BUS	SERIAL
I. COST								
1.	A		B,M	B,M		2	2	1
	B		B	B	S	2	4	1
2.	A		S	B	M	1	2	4
	B		B	M	S	2	4	1
	C		B	M	S	2	4	1
3.	A		M	B,S	B,S	4	2	2
	B		M	S	B	4	1	2
	C							
		1	M	B	S	4	2	1
		2	M	B	S	4	2	1
		3	S	B,M	B,M	2	2	4
		4	S	B	M	1	2	4
	D		S	B,M	B,M	2	2	4
4.	A		B	M	S	2	4	1
	B		B	M	S	2	4	1
	C		B	M	S	2	4	1
	D	(1)	B	M	S	2	4	1
	D	(2)	B	M	S	2	4	1
5.	A		B	S	M	1	4	2
	B		B	M	S	2	4	1
	C		B	M	S	2	4	1
6.	A		B	M	S	2	4	1
	B		B	M	S	2	4	1
	C		B,M	B,M	S	2	2	1
	D		M	S	B	4	1	2
	E		B	M	S	2	4	1
7.	A		M	S	B	4	1	2
	B.		S	M	B	2	1	4
	C		S	M	B	2	1	4
	D		S	B	M	1	2	4
COST TOTALS						66	81	55

Table G-1 (Cont).

	CHOICE			WEIGHTED SCORES		
	BEST	MID.	WORST	MATRIX	BUS	SERIAL
II. RELIABILITY						
1	B	S	M	1	4	2
2	S	M	B	2	1	4
3	M	B,S	B,S	4	2	2
4 (A)	B,M	B,M	S	2	2	1
4 (B)		B,M,S		2	2	2
5	B	M	S	2	4	1
RELIABILITY TOTAL				13	15	12
III. PERFORMANCE						
1.	M	B	S	4	2	1
2.	B		M,S	1	4	1
PERFORMANCE TOTAL				5	6	2

Bus — Timing and Control Module (2 4x7" cards/0-200 words of microcode) -3 BEU's (1 4x7" card each).

Matrix — Matrix switch and controller 2-3 4x7":cards/200-1000 words of microcode).

Serial — Technical and process control (1 minicomputer/10,000-40,000 lines of assembly language). 3 MIU's (5 4x7" cards each) for technical and process control and 2 hardware modules, 2 repeaters (2 4x7" cards each), 6 taps, 0 amplifiers.

B. What is needed to support each additional module? (See also 3A)

Bus — 1 BEU for each hardware module (complexity (1-3 4x7" cards) depends on module)/ 0-10 words of microcode in the control module for each vector interrupt that it has to generate, I/O support in other modules to move data to/from the additional module.

Matrix — One MIC per hardware module until all ports (16) are used. Hardware growth than halts unless a multiplexer or submatrix module is defined. Potentially the addition of any module (hardware or software) can require I/O support from other modules and microcode support in all MIC's. These requirements can be minimized by careful definition of software module interfaces and the use of generalized I/O routines in the MIC's.

Serial — 1 MIU and 2 taps per hardware module, 2 amplifiers per X taps and/or Y ft (20 dB loss), 1 repeater per data channel if additional bandwidth is required. Technical control must be informed of the characteristics of each new hardware module and process control must be informed of the configuration and procedure for all new system tasks.

2. Flexibility

A. How well are widely varying performance requirements handled?

Bus — Addition of new modules limited only by bandwidth and addressability. (Bus buffers must be added for groups of 100 modules.)

Matrix — Addition of new modules limited by bandwidth and severely limited by number of available ports. The limited number of connections makes it undesirable to utilize a port for a low speed device, but multiplexing multiple low speed devices onto a single port causes other complications.

Serial — Addition of new modules limited only by bandwidth (addressing capability is essentially infinite for signal processing).

B. Is it easy to go from the general to the specific?

Bus – A hierarchy of busses may be easily defined. This can be used to increase bandwidth and addressability.

Matrix – Assuming "smarts" are distributed in using modules, this is not too difficult. The design process is aided by concurrent applications design. A hierarchy of matrices and sub-matrices can be defined in extend addressability.

Serial – No. A good understanding of Technical and Process Control is necessary to achieve efficient interaction of the module with them. The present description of the software used to implement technical and process control is sketchy at best, but indications are that it is very general and very complex.

C. Is the specific still general relative to the class of problems?

Bus – Yes, if a hierarchy is used. Also, most microprocessors use an internal bus structure. This says that one may proceed from general to specific all the way down to this level. Note that in doing this, the details of control of busses at the different levels may be different. This increases the burden of documentation but allows existing modules which use different control mechanisms to be incorporated into a system. It also allows a "new and improved" module which uses a different control mechanism to be incorporated into an existing system.

Matrix – To a limited extent. This feature can be strengthened by the use of: matrix/ submatrix techniques, programmable modules, and general purpose I/O routines in the MIC's.

Serial – No. This system seems to go from the general solution of all the problems in the universe to the specific solution of a particular case in one full swoop.

3. Adaptable within a Configuration

A. How is growth in hardware or software achieved? (New design or repetition?)

Bus – Growth is easy and cost nearly linear. Slight discontinuities and hard limits as noted in 2A and 3C. Mechanical design may impose additional limits. When a hardware/software module is added, it is usually necessary for other modules to provide I/O support. Costs for I/O support approach a square law relationship for the worst case where all modules intercommunicate.

Matrix – If unused ports are defined in the original configuration, growth is easy until the ports are all used. Hardware growth within the configuration then ceases. I/O support costs will be moderate and nearly linear in a well designed matrix system. (See also 1B).

Serial — Growth is easy and cost nearly linear. Slight discontinuities occur when amplifiers must be added to gain data channel bandwidth. Costs for I/O support are nearly linear since I/O support is provided by process control on a per task basis. An exception to this is the case where adding a new module allows the system to do many new tasks.

B. How can field changes be tested and implemented?

Bus — Field changes to the hardware are reasonably easy as long as the mechanical design provides easy access to the bus. When a new module is added, a malfunction or design error in the new module may make the entire system inoperative. This risk is reduced by using a standard circuit for the BEU. Field changes to the software carry a similar risk of tying up the bus.

Matrix — Addition of a hardware module is easy if a port is available. The addition of software modules or replacement of a hardware module is also easy. The risk involved in such changes is very low since the matrix provides isolation between its ports.

Serial — Field changes are relatively easy. System down time for the addition of a hardware module is dependent on the ease of installing a tap. Since the taps provide isolation from the serial line and the operation of the MIU's can be verified prior to installation, additional down time is unlikely. A hardware/software module can be replaced with the system only losing the ability to perform tasks which require that module.

C. What limits does the design place on upper signaling rates, response times, dynamic range, memory or device addressability, etc.?

These performance related items affect cost in that they allow an assessment of the risk of exceeding a performance limit when one attempts to upgrade a system. Their cost impact arises at design/production time with the inclusion of the requisite performance margins.

	BUS	MATRIX	SERIAL
Data Rate	Mid	Best	Worst
Response Time	Best	Mid	Worst
Dynamic Range	Mid	Mid	Best
Addressability	Mid	Worst	Best

The bus has a cycle time of about 250ns and uses a 16 bit word. This gives a 64 MHz data rate. The matrix supports a 16 MHz data rate on each data line. Assuming eight pairs of ports simultaneously conversing and a 3% overhead for control characters gives a 124 MHz total

data rate. The serial system has an effective data rate of 7.5 MHz per FDM channel. This limit is a soft limit in that more FDM channels can be added at significant expense.

The response time of the bus is simply its cycle time (250ns). The turn around time of the matrix is 2.5 us. The turn around time of the serial system is at least 238 us.

The dynamic range of the bus is fixed by its 16 bit data word. Multiple words may be used to extend this, but such extension is not trivial. The matrix is oriented toward the use of eight bit characters, but these are easily manipulated in any multiple of eight bits. The serial system uses a 256 bit block which in most cases would be subdivided into words of appropriate length.

The bus uses a 16 bit address. The matrix can directly address only 16 ports (4 bits), memory management techniques are, therefore, mandatory in this sytem. The serial system has 65,536 time slots (16 bits) and each slot contains a 34 bit tag for an effective 50 bit addressing capability.

D. Can the design accommodate technology change in signaling techniques, filtering techniques, A/D dynamic range, etc.? (What is the probability of such change?)

Bus — Yes. The burden of change is on the modules.

Matrix — Yes. The burden of change is on the modules.

Serial — Most technology changes would affect only individual modules. Advances in the area of fiberoptics couid make the serial communication technique much more attractive.

4. Ease of Design

A. What hardware/software exists that is similar or identical to the required modules?

Bus — Much hardware exists which is very similar to that required. The BEU functions need only to be repackaged. Much software for bus-oriented machines also exists.

Matrix — Matrix systems are not common, but they are relatively easy to work with at the conceptual level.

Serial — Similar hardware is not well known. Extensive executive software is required.

B. Can "standard" components/languages be used?

Bus — Yes. Bus interfaces which are implemented with small scale integrated circuits already exist. Medium scale integrated circuits which now exist may be used to improve this interface. The bus architecture has no specialized language requirements.

Matrix — Yes. The matrix controller and interface units are implemented with microprocessors. The language used in this system must be able to generate efficient microcode.

Serial — Yes. This system uses a mixture of standard logic elements, standard CATV elements, and standard minicomputers. To improve cost, custom large scale integration should be considered for the interface units. This architecture has a stronger requirement for a job control or process control language than the other two do.

C. Are both the hardware and software natural to use or does one have to be continually "clever"?

Bus — Better than average. Cleverness is required only if bus bandwidth and/or software execution time limits are approached. The hardware module interface is extremely simple. A similarly simple software module interface can be used. The module designer/programmer needs to know very little about the rest of the system. As is always the case, it is highly desirable that at least one person on the project understand the entire system to insure efficient operation.

Matrix — Average design difficulty. Module design requires some knowledge of the system.

Serial — If the latency in moving data and getting a positive response can be tolerated, the system is reasonably natural to use. Good executive software is the key to this factor. For signal processing the latency is unfortunately long. This would require continual cleverness and record-keeping to overcome. Module design requires extensive knowledge of the system including executive software.

D. Can it be tested simply?
   1. Can it be tested incrementally?
   2. Can it be easily prototyped and/or simulated?

   Bus — 1. Yes.
   2. Yes.

   Matrix — 1. Yes. Minimal test configuration is somewhat large.
   2. Probably average.

Serial — 1. Yes. Minimal test configuration would be technical and process control, and one or two test modules. This is still a fairly large collection of hardware and software. Specialized executive test software would be very helpful if not essential.

2. Coarse simulation would be fairly easy. Prototyping would be a fairly major effort.

5. Ease of Documentation

A. Is a common interface used for software and/or hardware modules?

Bus — Can Be. As an example, a vectored interrupt can be processed by a dedicated hardware module or by a specific software module within a hardware module.

Matrix — A common hardware interface exists between the matrix switch and the MIC's. The MIC's are microcoded to meet the needs of the individual module. The module interface is therefore less standardized. Software interfaces can be standardized, but any interchange of software modules with hardware modules requires a change to the task keeper software.

Serial — A common hardware interface exists up to a point. Beyond this point, the "flexibility" exists to tailor the interface to the module. It is presumed that a similar situation would exist with the executive software. It is not readily apparent that the hardware and software modules could be easily interchanged.

B. To what degree is the hardware/software self-documenting?

Bus — BEU's are documented once and then used by all modules. High level language may be used.

Matrix — Hardware for a switch is largely application independent. Hence documentation is non-recurring. Control software for signal processing is not too bad. MIC hardware is documented only once. Microcode for individual MIC's should be very similar.

Serial — MIU's, bus repeaters, amplifiers, and taps are documented once and then used by all modules. Complexity of the MIU's is a disadvantage. "Cleverness" as described in 4C causes increased difficulty in documenting individual modules. Tailoring of the MIC's causes some problems. If the exec software is properly written, the user should be able to program process control functions in a very high level language. It may be more difficult to use a high level language to write the exec software

C. Did the original design light the way for future changes? This is necessary because personnel will usually be different between an initial design and the redesign event.

Bus — Better than average because of existing documentations and simplicity of architecture.

Matrix — Yes, if reasonable documentation support is maintained.

Serial — Can, but complexity of the system would require the original designer to be very conscientious in his documentation effort, particularly in the area of software documentation.

6. Manufacturability

A. Is it state-of-the-art? (Consideration must be given to the time of application, the necessity of the technique, and must include an effort to forecast industry trends.)

Bus — No.

Matrix — Not state-of-the-art. The elements that make up the matrix and its controller are benefitting from increased use of large scale integration.

Serial — No, but it might be if optical fiber techniques are used. The serial system should benefit from expected advances in the state-of-the-art in this area. The present high speed modems used are closer to being state-of-the-art than the elements used in the other systems.

B. Does the depth of the logic require long test sequences?

Bus — Test sequences are determined by modules. Almost no length is added to the test sequences as a result of the parallel bus architecture.

Matrix — Test may be complex — more so than a parallel structure at least. The larger amount of software required may be a benefit in larger production runs.

Serial — Serial interfaces tend to generate long test sequences.

C. Does "cleverness" of the design require high level test technicians?

Bus — Not usually. Certain types of failures on the bus are difficult to isolate. Data transactions on the bus can be difficult to observe. The use of logic state analyzers significantly reduces these problems.

Matrix — Normal test personnel requirements are adequate. The matrix doesn't have the [illegible] difficulties of the bus, but its circuitry is more complex.

Serial — Cleverness as described in 4C may require some high level test technicians. The mixture of logic type components and CATV type components found in this system will cause some additional difficulties.

D. Is it machine diagnosable? (Machine diagnosis may reduce but not eliminate the impact of the above two items.)

Bus — Machine diagnosis of module failures is possible. Machine diagnosis of bus failures is difficult and special provisions in the hardware are required to isolate such failures.

Matrix — Machine diagnosis of module failures is possible. Perhaps trouble at the module interface (and internal) level would require manual intervention. Monitoring of control line protocol and data line activity is "built in".

Serial — Machine diagnosis of module failures is possible. Some such features are "built in" to the system. Failures in the transmission medium should be easy to isolate manually. Failures in technical control could be difficult to find unless an independent console is attached directly to the minicomputer that provides the centralized portion of this function. Such a console and possibly additional I/O devices would allow extensive self-test routines to be implemented.

E. Can design phase tests be used to partially or wholly satisfy production test requirements?

Bus — Very little testing is required beyond the tests of the individual modules. A system level "exerciser" type of diagnositic should be easy to write for this system.

Matrix — Yes. Design and production test require module simulation. This is a design time requirement but is transferable to production. Production test requirements for MIC's and for the Matrix Controller may exceed those in the design phase.

Serial — Due to the complexity of the system, production testing will probably need to be much more extensive than that used in the design phase.

7. Maintainability

A. (Can fault isolation be easily achieved? (Busses make this difficult; also relative addressing of any kind, virtual machines, and any kind of multiplexing is less than optimum.)

Bus — Faults within modules that don't place a fault on the bus itself are easy to isolate. Faults on the bus can be very difficult to isolate.

Matrix - Fault isolation will be very good (rapid) to circuit-related (i.e., channel) failures. Control switch functions would be only average unless a "diagnostic" module were included in the system.

Serial – Most faults within modules are easy to isolate. This system makes it very unlikely for a module to place a fault on the serial bus. Most failures of amplifiers or repeaters can be quickly isolated from observing loss of signal or data. Transmission line failure can be easily isolated to a particular line segment. Failures in process and technical control may be difficult but most of these could be isolated with appropriate self-diagnostic programs.

B. Can test facilities be built in naturally?

Bus – Yes. A timeout for bus transactions is built into the control module. With medium scale integrated circuits that are now available, parity generation and checking can be added with no cost increase.

Matrix – Yes. All control and data lines are full duplex with the return path being used for control response and error control. Sync characters are transmitted constantly on idle control and data lines. For every character transmitted a character is also returned. The matrix controller constantly monitors operations for validity.

Serial – Yes. Some are included in the original definition of this architecture.

C. Cabling, connections, etc. Do these force unnatural mechanical designs which are hard to maintain?

Bus – The large number of wires in the parallel interface can cause difficulties.

Matrix – Cabling is better than average. The serial nature of the communications and the lower data rate (i.e., data rate is a per-channel phenomena – not a total) yield simpler connections.

Serial – Cabling is very simple. Special attention must be paid to the coax connectors.

D. Can the mechanical design be transferred easily? (This is a measure of the simplicity of connection.)

Bus – Yes, but the large number of wires can cause some problems.

Matrix – Mechanical design for expandability is easy only within limits. Extra channels may easily be added but the basic size of the switch cannot be changed in existing equipment. New designs could accommodate larger switches without much design time.

Serial – Yes. Special attention must be paid to the coax connectors.

III. Relaibility

1. Ascertain MTBF

   The parallel system, using estimated part counts and number of connections, should have the best MTBF. Serial is ranked second and Matrix third. Matrix has a poor showing here because of the poor estimated failure rate of microprocessors. (The Matrix system studied uses one microprocessor per interface unit plus one in the Matrix Switch Module. The serial system requires a large microprocessor or minicomputer for control, while the parallel has a much simpler mechanism).

2. Ascertain the number of single point failure locations

   Because of the distributed nature of the control, and four circuit connections to the central bus per interface unit, the serial system rates as having fewest single point failure locations. The matrix, because of the complexity of the central Matrix Switch Module, ranks below the serial, while the parallel bus system, with its relatively high number of lines, rates lowest. It should also be noted that this category is rated the most important in the reliability section.

3. Ascertain the types of failure modes

   Failure modes must be identified in terms of their sytem impact (percent loss of function). Critical sections must be analyzed for failure detectability.

   This category is rated as the second most important in this section. However, the failure modes for all systems appear to be identical (failure to release, failure to connect, wrong connection, etc.). The Matrix system has a better detectability because of the one-to-one connection path.

4. Is the design pushed to technology limits?

   A. Timing Constraints

      The serial system, becuase of the high speed, has the only timing problem forseen.

   B. Environmental Sensitivity

      No system appears to have any particular advantage or disadvantage.

5. What is the maturity of the technology itself?

   The parallel system is the most used system, with most, if not all, elements of the system included in MSI and LSI parts which are available off the shelf. The Matrix system does not have the level of

integration of the parallel system, but is completely digital, and contains no high risk developmental areas. The serial system, rated lowest in the category, will require the development of the data modems.

IV. Performance

To determine the Data Processing capability each architecture was considered in terms of maximum point to point transfer rate.

1. Point to Point Transfer Rate.

Parallel Bus – The transfer rate of the Parallel Bus approaches a 4 MHz transfer rate if no time is allowed for data set up in the modules connected to it. This can be defined as 240 nsec + Module set up or access time. If C-mos modules or other slow devices are used in the module the bus rate drops significantly unless high speed buffering is provided.

Each transfer on the bus consists of a parallel 16 bit word and a 16 bit address (this transfer includes parity if so desired).

Matrix – The transfer rate of the matrix was investigated using two approaches. The minimum hardware approach uses a MOS UARTS which have an upper rate of 56 KHz. This yields a transfer rate of 6.1K bytes or 3K 16 bit-word transfers. The other extreme uses a Manchester coded serial stream at 16 MHz. This yields a 2M byte transfer rate of 1 MHz 16 bit word transfers per data path. Up to eight data paths may be active at once in the 16 port matrix.

This approach yields a high throughput rate at the expense of higher hardware cost. The rating was made using the higher transfer rate.

Serial Bus – The serial bus was also investigated using two approaches. The first assumed only two frequencies available in the FDM system. One frequency would be used for data and the other for control. With a 14.8 MHz channel, the transfer rate is 925K 16 bit words. With 10 channels, this rate is 925M 16 bit words. The second approach is very costly in hardware but it should be noted that growth from 925 KHz to 9.25 MHz increases linearly in cost with increased performance. This rate is slightly lower when the overhead is added (7.54 MHz).

The results of the comparison in terms a data transfer rate show the Matrix first, The Parallel Bus second and the serial FDM/TDM third.

The second attribute to investigate in Performance is the time to make or break a connection between modules. This is one measure of the efficiency of the control mechanism in the architecture.

2. Control Requirements.

Parallel Bus – No penalty in throughput rate is suffered due to the control scheme of the parallel bus due to the "Next Master" and "Current Master" states of the bus. Due to the non-synchronous nature of the data transfers and the single communication path present, any signal processing implementation using the parallel bus must insure proper ordering of module priority on the bus. Also, care must be taken to insure any module with slow access does not interfere with a module with high data transfer rates.

Matrix – The matrix system was again considered using two approaches. With the UART approach, the time to disconnect one connection and make the next connection (turn-around time) is 571 msec which appears to be not useable in signal processing without large amounts of module buffering. For the 16 MHz coded scheme, this time reduces to 2.5 usec, but severe loading on the control processor is forced. This load appears to be greater than can be handled by even a fast processor.

Without special hardware the UART, approach provides a high load on the control processor. This load can be reduced by implementing a scheme in which the hardware filters all "sync" characters and allows only essential control characters to reach the processor interface.

To find a matrix control speed which satisfies the signal processing requirements requires investigation on a case-by-case basis.

Serial Bus – The Serial Bus has a turn-around time at best of 237.8 usec assuming 33,640 control sequences per second and eight modules in the system. The matrix defined allows for only 1,462 control sequences per second or a 5.4 ms turn-around time. Neither of these times appear to be useable in a signal processing system. It should be noted that in the 10 channel system this problem is not present since all connections can be established at once.

The results of the comparison in terms of Control Requirements show the Parallel Bus first. The Serial FDM/TDX Bus, and the Matrix do not appear to satisfy signal processing requirements.

APPENDIX H

ARCHITECTURE SELECTION FOR SWITCHING SYSTEMS

I. Summary

A functional layout for message switching for each of the three architectures has been formulated based on the descriptions in Appendix A. The following general assumptions have been made to allow a valid comparison among the three. The functional task modules necessary in the message switch are assumed to be designed such that they are independent of their interconnection. Any interconnection dependencies will be resolved in the module interface units, i.e., BEU, MIU, MIC as appropriate.

The items that must be evaluated in the selection of one of the three architectures that are unique to their respective architecture are listed below:

Architecture	Interface Module	Interconnect Scheme	System Control
Serial	MIU	Coax bus prior	Tech Control, Repeaters, and Process Control
Matrix	MIC	Matrix hardware	System Control
Parallel Bus	BEU	Thirty four parallel lines	Timing and Control

Table H-1 summarizes the results of these analyses in raw score form. Weighted score and best worst comparisons are given in figures H-1 and H-2, respectively.

II. Architecture Comparison by Cost

1. Required Hardware/Software
   A. Needed to support minimum configuration. The minimum configuration for a switching system would be one CPU, one main memory, one line or trunk handler, and whatever support hardware and software that is required by the particular architecture. The minimum configuration considered to be of significant practical interest, however, includes one CPU, one main memory, three line handlers, and one trunk handler. It is support for this configuration that is considered below.

      Parallel Bus

      6 BEU's (2 4" x 7" circuit cards each)

Table II-1.

		CHOICE			WEIGHTED SCORES		
		BEST	MID.	WORST	MATRIX	BUS	SERIAL
I. COST							
1.	A	B	M	S	2	4	1
	B	B	S	M	1	4	2
2.	A	S	B	M	1	2	4
	B	B	M	S	2	4	1
	C	B	M	S	2	4	1
3.	A	B	S	M	1	4	2
	B	M	S	B	4	1	2
	C		B,M,S		2	2	2
4.	A	B	M	S	2	4	1
	B	B	M	S	2	4	1
	C	B	M	S	2	4	1
	D(1)	B	M	S	2	4	1
	D(2)	B	M	S	2	4	1
5.	A	B	S	M	1	4	2
	B	B	M	S	2	4	1
	C	B	M	S	2	4	1
6.	A	B	M	S	2	4	1
	B	B	M	S	2	4	1
	C	B,M	B,M	S	2	2	1
	D	M	S	B	4	1	2
	E	B	M	S	2	4	1
7.	A	M	S	B	4	1	2
	B	S	M	B	2	1	4
	C	S	M	B	2	1	4
	D	S	B	M	1	2	4
COST TOTALS					51	77	44
II. RELIABILITY							
	1	B	M,S	M,S	2	4	2
	2	B	M	S	4	1	2
	3	M	S	B	4	1	2
	4(A)	B	M	S	2	4	1
	4(B)	B	M	S	2	4	1

Table H-1 (Cont).

	CHOICE			WEIGHTED SCORES		
	BEST	MID.	WORST	MATRIX	BUS	SERIAL
II. RELIABILITY (Cont.)						
5	B	M	S	2	4	1
RELIABILITY TOTAL				16	18	9
III. PERFORMANCE						
1.	M	B	S	4	2	1
2.	B	S	M	1	4	2
PERFORMANCE TOTAL				5	6	3

Matrix

7 MIC's (3 4" x 7" circuit boards/500-1200 words of microcode each)

1 Matrix switch and controller (3 4" x 7" circuit boards/200-800 words of microcode)

Serial Bus

7 MIU's (5 4" x 7" circuit boards each)

2 repeaters (2 4" x 7" circuit boards each)

14 taps

1 technical and process controller (1 minicomputer/8K—20K lines of assembly language)

B. Needed to support each additional module

Parallel Bus - To add one line handler or trunk handler to the system requires the addition of one BEU (2 - 4 x 7" circuit boards). Slight additions must be made to the CPU software to extend such things as routing tables, interrupt handlers, etc. This addition of modules can go on almost indefinitely until traffic causes congestion on the bus, or the CPU or memory become overloaded. Memory size can be increased with no other additional hardware and only a minor change to the memory allocation routines in the CPU. Alternatively, an additional memory module and BEU can be attached to the bus. This approach also increases available memory bandwidth if the bus bandwidth exceeds that of a single memory module.

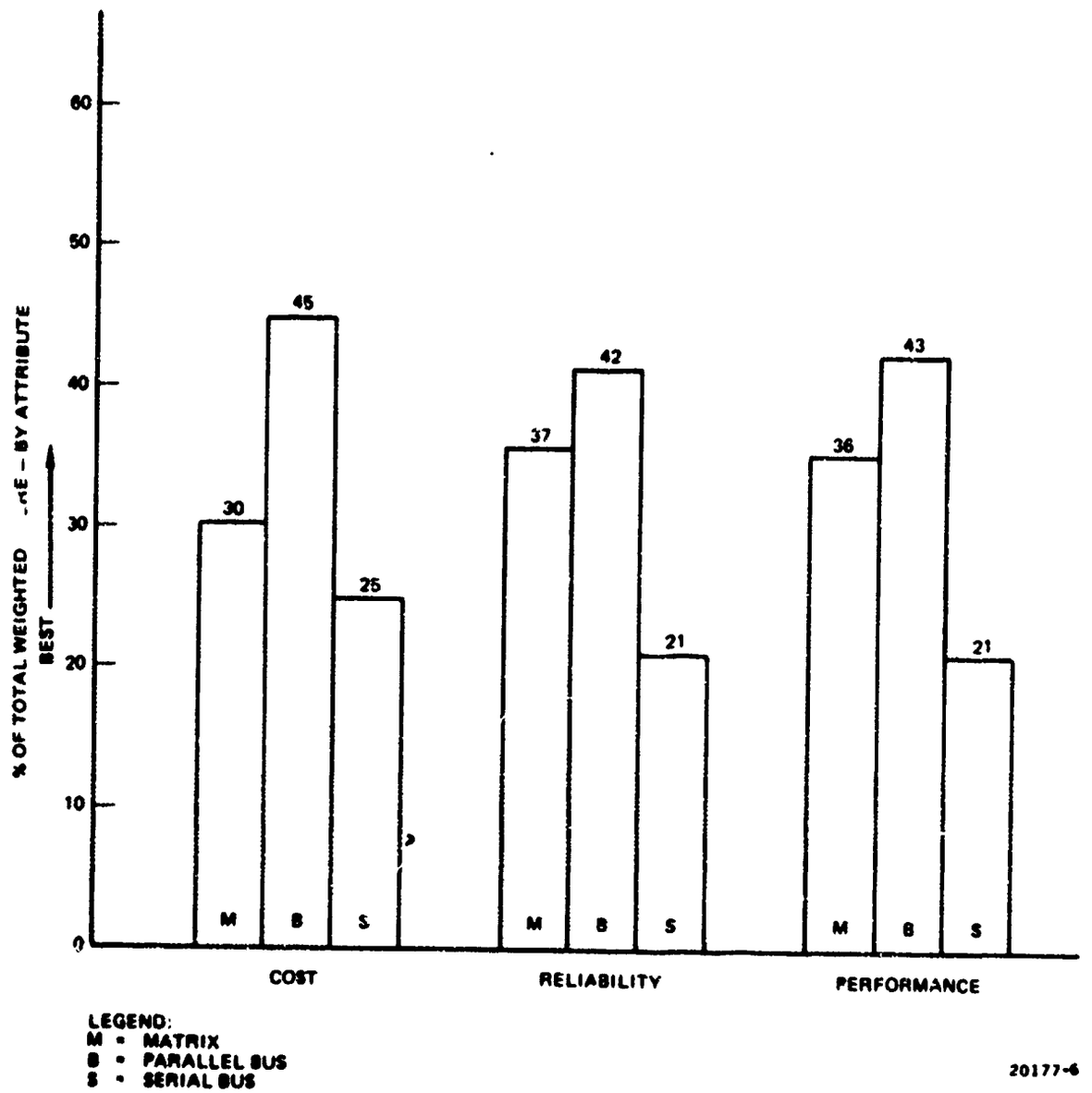


Figure H-1. Percent Weighted Score.

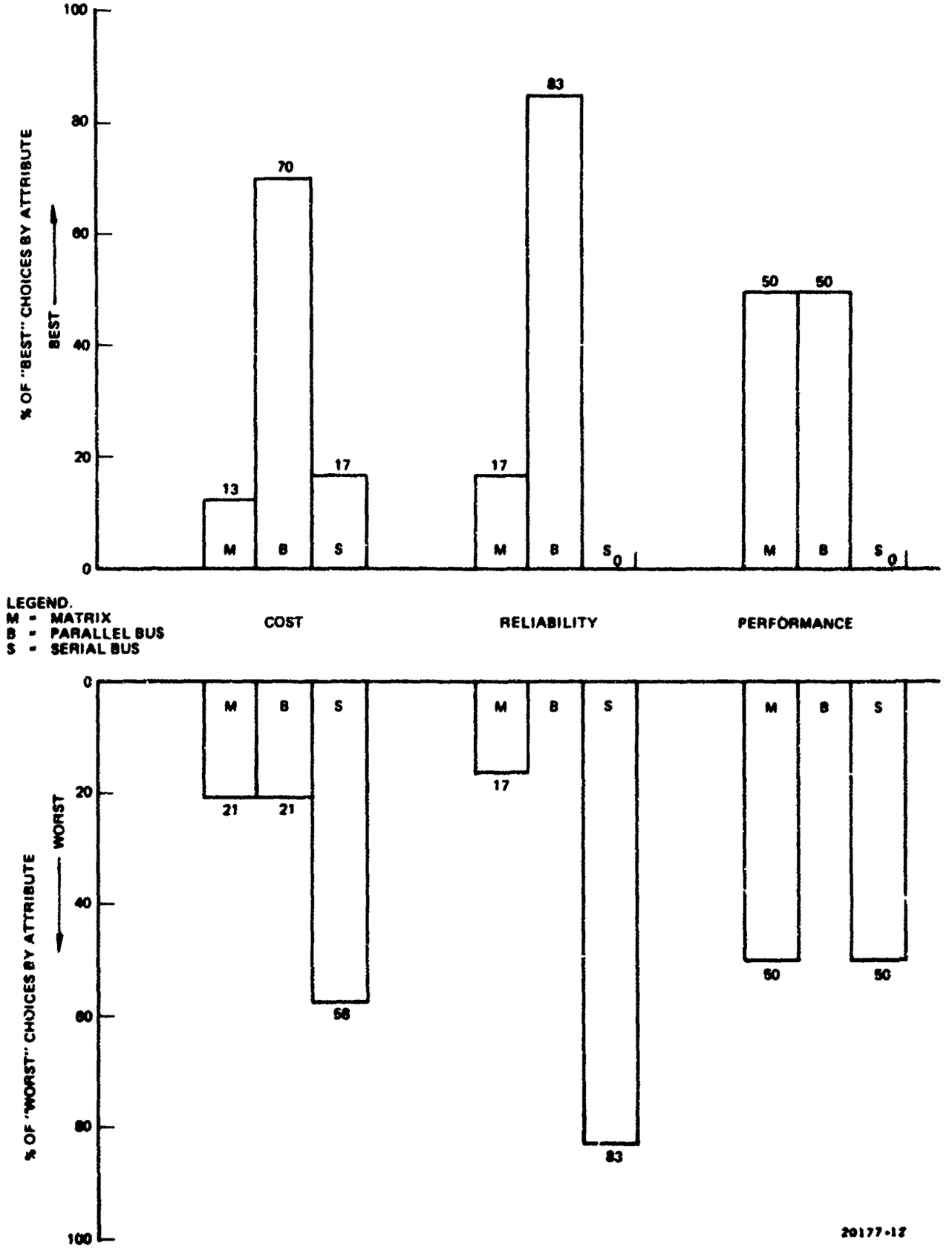


Figure H-2. Best/Worst Comparison.

Adding a CPU to the system requires an additional BEU. Software in the original CPU may be repartitioned or duplicated. Assuming that the original software was modular leads to the conclusion that repartitioning is easy. If duplicate code is to be used, the original design must have provided for such things as shared access to tables and dynamic task queing.

Matrix - To add a line handler to the system requires the addition of an MIC module (3 - 4 x 7" circuit boards) provided the matrix switch was designed for the additional paths. This addition can continue only as long as spare paths, which were originally designed in, are available. Multiplexing hardware could have been designed in to allow fewer paths to handle more modules. This would have improved expandibility but would require more hardware and would have destroyed the non-blocking features of the matrix. Expandibility could also be improved by utilizing a network of sub-matrices to which other sub-matrix switches could be added.

If software is handled properly within the MIC during original design, software modifications can be minimum or none at all when MICs are added. Software in the matrix controller would be written to support a fully implemented matrix and would require no modification. Addition of CPU's and memory to this system is subject to the same considerations as for the parallel bus.

Serial Bus - To add a line handler to the system requires the addition of an MIU, two bus taps, and modifications to the control software. Technical control must be informed of the characteristics of each new hardware module and process control must be informed of the configuration and procedures for all new system tasks. Adding a CPU to this system is not significantly different than adding a line handler since process control takes care of task scheduling. Memory may be added with or without additional MIU's and taps.

2. Flexibility

The rationale for the selection of the architecture based on flexibility appears to be identical whether the application is for signal processing or for message switching.

A. How well are widely varying performance requirements handled?

Bus - Addition of new modules limited only by bandwidth and addressability. (Bus buffers must be added for groups of 100 modules.)

Matrix - Addition of new modules limited by bandwidth and severely limited by number of available ports. The limited number of connections makes it undesirable to utilize a port for a low speed device, but multiplexing multiple low speed devices onto a single port causes other complications.

Serial - Addition of new modules limited only by bandwidth (addressing capability is essentially infinite for switching).

B. Is it easy to go from the general to the specific?

Bus - A heirarchy of busses may be easily defined. This can be used to increase bandwidth and addressability.

Matrix - Assuming "smarts" are distributed in using modules, this is not too difficult. The design process is aided by concurrent applications design. A heirarchy of matrices and submatrices can be defined to extend addressability.

Serial - No. A good understanding of Technical and Process Control is necessary to achieve efficient interaction of the module with them. The present description of the software used to implement technical and process control is sketchy at best, but indications are that it is very general and very complex.

C. Is the specific still general relative to the class of problems?

Bus - Yes, if a heirarchy is used. Also, most microprocessors use an internal bus structure. This says that one may proceed from general to specific all the way down to this level. Note that in doing this the details of control of busses at the different levels may be different. This increases the burden of documentation but allows existing modules which use different control mechanisms to be incorporated into a system. It also allows a "new and improved" module which uses a different control mechanism to be incorporated into an existing system.

Matrix - To a limited extent. This feature can be strengthened by the use of: matrix/submatrix techniques, programmable modules, and general purpose I/O routines in the MICs.

Serial - No. This system seems to go from the general solution of all the problems in the universe to the specific solution of a particular case in one fell swoop.

3. Adaptable Within a Configuration

A. Growth (see also section 2A)

This section will address only the addition of line handlers to the system. If the original design provided for increases in the size of memory or number of CPU's, then the cost to support such growth will be either negligable or roughly equivalent to that for adding a line handler. If such provisions were not made in the original design, then growth in these areas is not growth "within the configuration."

Bus - Can be expanded by adding a BEU for each line handler and by adding the addresses of the added devices to the appropriate tables in the CPU software.

Matrix - Can be expanded by adding an MIC for each line handler, if unused matrix ports were defined in the original configuration. Once the ports are all used, growth within the configuration ceases. Since the size of the matrix is known at design time, the CPU software would probably be written to include all possible device addresses in its original tables. Therefore, no software changes would be needed for growth within the configuration.

Serial - Can be expanded by adding an MIU and two taps for each line handler. CPU tables must be updated and process and technical control must be informed of the existance of the new devices.

B. How can field changes be tested and implemented?

Bus - Field changes to the hardware are reasonably easy as long as the mechanical design provides easy access to the bus. When a new module is added, a malfunction or design error in the new module may make the entire system inoperative. This risk is reduced by using a standard circuit for BEU. Field changes to the software carry a similar risk of tying up the bus.

Matrix - Addition of a hardware module is easy if a port is available. The addition of software modules or replacement of a hardware module is also easy. The risk involved in such changes is very low since the matrix provides isolation between its ports.

Serial - Field changes are relatively easy. System down time for the addition of a hardware module is dependent on the ease of installing a tap. Since the taps provide isolation from the serial line and the operation of the MIU's can be verified prior to installation, additional down time is unlikely. A hardware/software module can be replaced with the system only losing the ability to perform tasks which require that module.

C. Can the design accommodate technology change in signaling techniques, filtering techniques, etc.? (What is the probability of such change?)

Bus - Yes - Burden of change is on the module.

Matrix - Yes - Burden of change is on the module.

Serial - Most technology changes would affect only individual modules. Advances in the area of fiberoptics could make the serial communication technique much more attractive.

4. Ease of Design

The message switching selection rationale is identical to that for signal processing.

A. What hardware/software exists that is similar or identical to the required modules?

Bus - Much hardware exists which is very similar to that required. The BEU functions need only to be repackaged. Much software for bus-oriented machines also exists.

Matrix - Matrix systems are not common, but they are relatively easy to work with at the conceptual level.

Serial - Similar hardware is not well known. Extensive executive software is required.

B. Can "standard" components/languages be used?

Bus - Yes. Bus interfaces which are implemented with small scale integrated circuits already exist. Medium scale integrated circuits which now exist may be used to improve this interface. The bus architecture has no specialized language requirements.

Matrix - Yes. The matrix controller and interface units are implemented with microprocessors. The language used in this system must be able to generate efficient microcode.

Serial - Yes. This system uses a mixture of standard logic elements, standard CATV elements, and standard minicomputers. To improve cost, custom large scale integration should be considered for the interface units. This architecture has a stronger requirement for a job control or process control language than the other two do.

C. Are both the hardware and software natural to use or does one have to continually be "clever"?

Bus - Better than average. Cleverness is required only if bus bandwidth and/or software execution time limits are approached. The hardware module interface is extremely simple. A similarly simple software module interface can be used. The module designer/programmer needs to know very little about the rest of the system. As is always the case, it is highly desirable that at least one person on the project understand the entire system to insure efficient operation.

Matrix - Average design difficulty. Module design requires some knowledge of the system.

Serial - If the latency in moving data and getting a positive response can be tolerated, the system is reasonably natural to use. Good executive software is the key to this factor. Module design requires extensive knowledge of the system including executive software.

D. Can it be tested simply?

1. Can it be tested incrementally?
2. Can it be easily prototyped and/or simulated?

Bus - 1. Yes
- 2. Yes

Matrix - 1. Yes. Minimal test configuration is somewhat large.
- 2. Probably average.

Serial - 1. Yes. Minimal test configuration would be technical and process control and one or two test modules. This is still a fairly large collection of hardware and software. Specialized executive test software would be very helpful if not essential.
- 2. Coarse simulation would be fairly easy. Prototyping would be a fairly major effort.

5. Ease of Documentation

A. Common Interface for software and/or hardware modules?

Bus - Can Be. As an example, a vectored interrupt can be processed by a dedicated hardware module or by a specific software module within a hardware module.

Matrix - A common hardware interface exists between the matrix switch and the MIC's. The MIC's are microcoded to meet the needs of the individual module. The module interface is therefore less standardized. Software interfaces to the line handlers are standardized, but it is not readily apparent that the hardware and software modules could be easily interchanged.

Serial - A common hardware interface exists up to a point. Beyond this point, the "flexibility" exists to tailor the interface to the module. It is presumed that a similar situation would exist with the executive software. It is not readily apparent that the hardware and software modules could be easily interchanged.

B. Degree of self-documentation of the hardware/software

Parallel Bus - The BEU's common boards are documented once and then used by all modules. Each unique board must be documented. The BEU mechanical packaging can be documented once for all BEU's.

Matrix - The MIC common boards are documented once for all MIC's. The MIC mechanical packaging can also be documented once for all MIC's. The system control module must have documentation for each of its circuit boards and its mechanical packaging. Software for the system control module and MIC's must be documented. The software for the individual line handlers is identical or nearly identical.

Serial Bus - The MIU common boards and the MIU mechanical packaging are documented once for all MIU's. The technical and process control minicomputer and associated software must be documented separately. The bus repeater and taps can be documented once each and repeated for all others.

C. Did the original design light the way for future changes? This is necessary because personnel will usually be different between an initial design and the redesign event.

Bus - Better than average because of existing documentation and simplicity of architecture.

Matrix - Yes, if reasonable documentation support is maintained.

Serial - Can, but complexity of the system would require the original designer to be very conscientious in his documentation effort, particularly in the area of software documentation.

6. Manufacturability

A. State of the Art

Parallel Bus - The parallel bus does not utilize new techniques or components. The simplicity of the BCU's and the low quantity of circuit board types make it easy to manufacture. The state-of-the-art could be improved by high usage of LSI's. All parts are readily available off-the-shelf.

Matrix - The matrix also is not pushing the state-of-the-art. The elements that make up the matrix and its controller are benefitting from increased use of LSI. All components are off-the-shelf. Manufacturability is not as easy as the parallel bus because of the increased circuit board types and modules.

Serial Bus - The serial bus is probably closer to the state-of-the-art than any of the three architectures. If fiber-optic buses were utilized, the architecture would be pushing the state-of-the-art. Manufacturability is more difficult than the other two architectures because its high quantity of non-identical modules and circuit board types.

B. Does the depth of the logic require long test sequences?

Bus - Test sequences are determined by modules. Almost no length is added to the test sequences as a result of the parallel bus architecture.

Matrix - Test may be complex - more so that a parallel structure at least. The larger amount of software required may be a benefit in larger production runs.

Serial - Serial interfaces tend to generate long test sequences.

C. Does "cleverness" of the design require high-level test technicians?

Bus - Not usually. Certain types of failures on the bus are difficult to isolate. Data transactions on the bus can be difficult to observe. The use of logic state analyzers significantly reduces these problems.

Matrix - Normal test personnel requirements are adequate.

Serial - Cleverness as described in 4C may require some high level test technicians. The mixture of logic type components and CATV type components found in this system will cause some additional difficulties.

D. Is it machine diagnosable? (Machine diagnosis may reduce but not eliminate the impact of the above two items.)

Bus - Machine diagnosis of module failures is possible. Machine diagnosis of bus failures is difficult and requires special provisions in the hardware to isolate such failures.

Matrix - Machine diagnosis of module failures is possible. Perhaps trouble at the module interface (and internal) level would require manual intervention. Monitoring of control line protocol and data line activity is "built in".

Serial - Machine diagnosis of module failures is possible. In fact, some such features are "built in" to the system. Failures in the transmission medium should be easy to isolate manually. Failures in technical control could be difficult to find unless an independent console is attached directly to the minicomputer that provides the centralized portion of this function. Such a console and possible additional I/O devices would allow extensive self-test routines to be implemented.

E. Can design phase tests be used to partially or wholly satisfy production test requirements?

Bus - Very little testing is required beyond the test of the individual modules A system level "exerciser" type of diagnostic should be easy to write for this system.

Matrix - Yes. Design and production tests require module simulation. This is a design time requirement but is transferable to production.

Serial - Due to the complexity of the system, production testing will probably need to be much more extensive than that used in the design phase.

7. Maintainability

The message switching selection rationale for maintainability is identical to that for signal processing.

A. Can fault isolation be easily achieved? (Busses make this difficult; also relative addressing of any kind, virtual machines, and any kind of multiplexing is less than optimum.)

Bus - Faults within modules that don't place a fault on the bus itself are easy to isolate. Faults on the bus can be very difficult to isolate.

Matrix - Fault isolation will be very good (rapid) to circuit-related (i.e., channel) failures. Control switch functions would be only average unless a "diagnostic" module were included in the system.

Serial - Most faults within modules are easy to isolate. This system makes it very unlikely for a module to place a fault on the serial bus. Most failures of amplifiers or repeaters can be quickly isolated from observing loss of signal or data. Transmission line failure can be easily isolated to a particular line segment. Failures in process and technical control may be difficult but most of these could be isolated with appropriate self-diagnostic programs.

B. Can test facilities be built in naturally?

Bus - Yes. A timeout for bus transactions is built into the control module. With medium scale integrated circuits that are now available, parity generation and checking can be added with no cost increase.

Matrix - Yes. All control and data lines are full duplex with the return path being used for control response and error control. Sync characters are transmitted constantly on idle control and data lines. For every character transmitted a character is also returned. The matrix controller constantly monotors operations for validity.

Serial - Yes. Some are included in the original definition of this architecture.

C. Cabling, connections, etc. Do these force unnatural mechanical designs which are hard to maintain?

Bus - The large number of wires in the parallel interface can cause difficulties.

Matrix - Cabling is better than average. The serial nature of the communications, and the lower data rate (i.e., data rate is a per-channel phenomena - not a total) yield simpler connections.

Serial - Cabling is very simple. Special attention must be paid to the coax connectors.

D. Can the mechanical design be transferred easily? (This is a measure of the simplicity of connection.)

Bus - Yes, but the large number of wires can cause some problems.

Matrix - Mechanical design for expandability is easy only within limits. Extra channels may easily be added but the basic size of the switch cannot be changed in existing equipment. New designs could accommodate larger switches without much design time.

Serial - Yes. Special attention must be paid to the coax connectors.

III. Reliability Comparison

1. MTBF

MTBF depends primarily on the types of parts used, the quantity of various types of parts, and the number of connections. The interconnected task modules are not included in an MTBF evaluation because these modules are constant regardless of the architecture selected. Only the interface modules, the interconnections between the interface modules, and the method of technical and system control, are unique to the architectures and are considered in the MTBF selection.

Interface Modules — The MIU (Serial) and the MIC (Matrix) interfaces are about equal in types and quantities of hardware. The BEU is much less complex than the others and has less hardware. Each MIC (Matrix) module contains a microprocessor for path control, and therefore is rated poor in MTBF.

Interconnect Scheme — The interconnection of the serial bus is two coax connectors per MIU which is the fewest quantity of connectors of the three. The Matrix requires an interconnect between each MIC and the matrix switch which is roughly equivalent to the serial bus quantity; however, the matrix has the additional connections of the matrix switch, itself. The parallel bus contains 34 wires, requiring 34 connections to each BEU; therefore should have the worst reliability of the three architectures as far as interconnects are concerned.

Technical and System Control — Technical and system control, including timing, is distributed within the task modules in the parallel bus architecture. Each line handler module does its own transferring and fetching of data via the BEU. The BEU is completely asynchronous. This technique requires less hardware in the BEU than would be required in the serial bus MIU or the matrix MIC.

It is the responsibility of the MIC in the matrix system to request connection to a specific port or to request disconnection from the current port. The actual establishment of connections is done by the matrix switch controller.

The serial bus MIU contains hardware for keeping track of time slot and address assignments, as well as bus synchronization. Further bus synchronization hardware is present in the bus repeater. This system has a minicomputer dedicated to the overall function of technical and process control.

2. Single Point Failure

Interconnect — Every wire connection is a potential single-point failure. Since the interconnect for the serial bus is two coax connections to each MIU, it will have the fewest single point failures.

The matrix has a group of control wires in addition to the data line that connects from each MIC to the matrix switch. Each of these connections represent failure points, but these are not necessarily complete system failures.

Since the parallel bus contains 34 wires that connect to every BEU, it has the highest number of points of failure. However, the potential for these particular failures is not as high as those described for the serial interconnections, because of the lower reliability of coax connectors.

Interface Modules — The interface modules in order of complexity and part count are: MIU (Serial), MIC (Matrix), and BEU (Parallel). Therefore, the potential for single point failure is in the same order.

Technical and System Control - The serial bus and the matrix contain more hardware and software modules for system control than does the parallel bus; i.e., the serial bus requires the bus repeater and technical control minicomputer for technical control of the bus. The matrix requires control and record keeping of the interconnect paths through the matrix switch. These additional modules give additional failure points. Therefore, the potential for single point failure due to control modules, highest to least, is: serial, matrix, parallel.

3. Types of Failure Modes

As in the signal processing section (Appendix G), the failure modes attributable to the architecture are the same for all three architectures: Failure to release a connection, failure to make a connection, making wrong connection, etc. The mechanisms that create the failure modes are different, with different failure rates, but the modes themselves remain the same. The detectability of some of the modes is better in the Matrix and Serial systems. The Matrix system has good detectability because of the one-to-one connection path. The Serial system has built in monitoring of time slot and identification assignments.

4. Technology Limits

a. Timing Constraints - None of the architectures appear to be pushing the technology limits. The serial bus, because of its high speed, bit-sync, slot-sync, and frame-sync requirements has the most severe timing constraints.

b. Environmental Sensitivity - The serial bus, because of its timing and circuit complexity, is most likely to be influenced by environmental extremes.

5. Maturity of Technology

Each of the three architectures can be realized with today's technology. The paraliel bus architecture seems to be the best understood and most used; therefore, it has the largest family of specialized components. The Serial Bus system is not widely used in localized switching processors.

IV. Performance Comparison

1. Data Processing Capability

Once the transmission path has been established (data transfer only), the matrix architecture and the serial bus are very close in data transfer rate with the parallel bus being slowest.

The matrix is the fastest of the three architectures because of its dedicated transmission paths.

Each device on the serial bus has its own bandwidth assigned; i.e., it has its own time slot(s) assigned. In a sense, these time slots can be considered parallel paths in that they are assigned and no contention for a particular assigned slot exists; however, there is a time delay involved in utilizing the slots. The transfer of data on the parallel bus architecture must be considered serial in the sense that only one device at a time can pass data on the bus.

2. Control Overhead

A. Establishing Interconnect - The matrix architecture requires more overhead in establishing the interconnects which partially offsets its advantage in data transfer. The serial bus requires a lesser amount of overhead in control of time slots. The parallel bus has minimal overhead since the time necessary to request and acquire the bus is overlapped with the preceeding data transfer.

B. Protocol - The matrix appears to require more protocol between processors and hardware blocks in establishing transmission paths than either the serial bus or the parallel bus. The serial bus and parallel bus are both relatively low in this type overhead.

C. System Overhead - The serial bus is very high in overhead bandwidth for maintaining system and bus synchronization. It also requires a high level of hardware for the synchronization and time slot counting. The parallel bus requires less control overhead than any of the other architectures, having transfer information contained in the parallel bus with the data.

APPENDIX I

SIGNAL PROCESSING MODULES

FUNCTIONAL MODULES OF A SIGNAL PROCESSING SYSTEM

The following sections are a group of modules defined for a strawman Signal Processing task chosen for use in the architecture simulation phase of this study. This communication mode is an encrypted, band-spread, antipodal, phase-continuous FSK communication mode with error detection and correction. Signal processing starts at the radio receiver IF frequency and ends with error corrected characters. Information flow is shown in Figure I-1. Message blocking, redundancy testing, message routing, etc., are not contained within this demodulator but are left for the next higher level user such as a switch or a radio operator. The signaling rates within this example are low, but it is felt a good understanding of the characteristics of the architecture modeled can be gained by using this example.

This group of module descriptions is necessarily slanted towards the parallel architecture. To apply these modules to the Matrix or Serial architecture will require a slight change in the control structure. This is caused due to the non-centralized control used in these architectures. Header words or control messages are all that is required in this conversion.

I.1 TIMING AND EXECUTIVE MODULE

The executive control of this signal processing example takes advantage of the fixed rate processing inherent in Signal Processing Algorithms by pipelining the independent operations. Centralized control simplifies the problem of controlling the modules within the unit and is implemented by allocating tasks based on a central timing chain. Allocation of all buffers within the unit is under the control of the executive as well as status monitoring, error reporting and diagnostic control. Figure I.1-1 is a block diagram of the elements within this module. Operation of the Task Scheduling is initiated by an interrupt from the central timing chain. Tasks are then passed to the Task Dispatcher via the Task Queue. The Task Dispatcher is responsible for initiating and monitoring all algorithms within the signal processor.

I.1.1 Module I/O

There are three interfaces to this module. They include: a) Direct Timing for all signals higher in frequency than normally controlled by a processor such as the 30 kHz squarewave sent to the IF Pre-processing Module, etc., b) Vector Interrupts used to control processing modules, and c) Programmed interface used to collect status or send control information.

A. Direct Interface
   1. Reference Timing - Frequency reference used to drive the central timing chains.
   2. 30 kHz Squarewave - Sent to the IF Pre-processing Module.
   3. 1600 Hz (or 20 kHz) Squarewave - Sent to the KG.

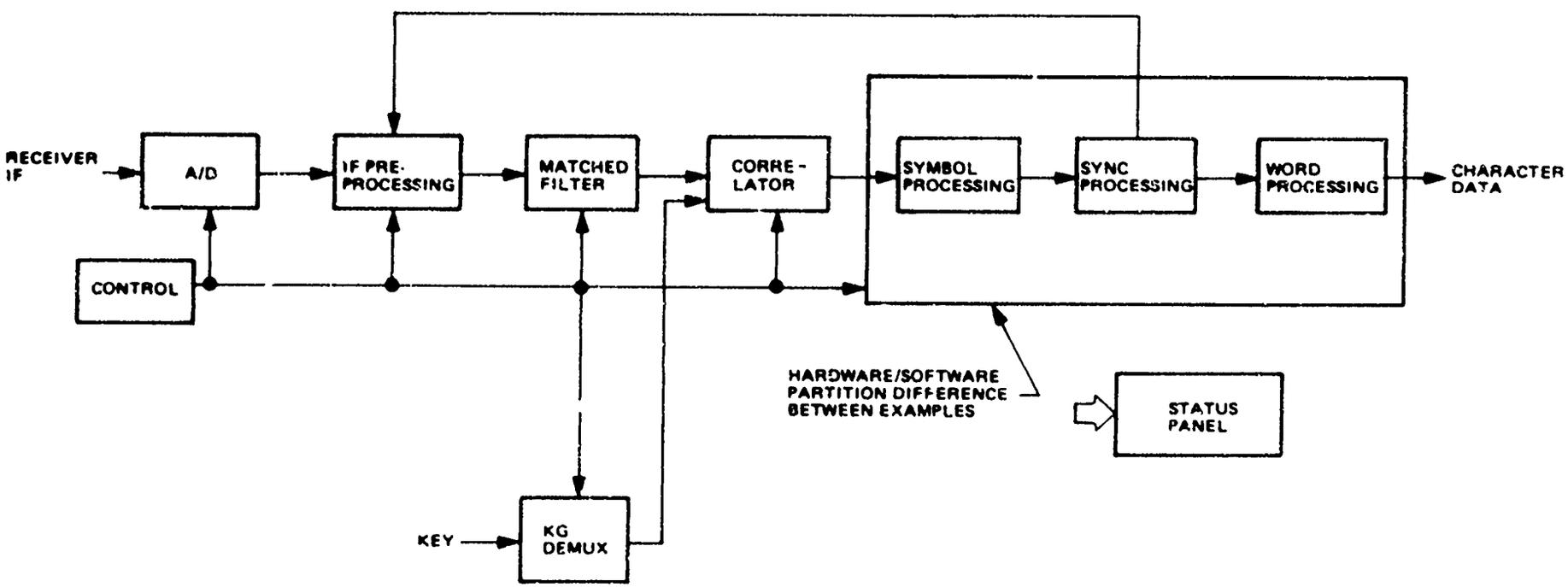


Figure 1-1. Information Flow Diagram for Signal Processing Modules.

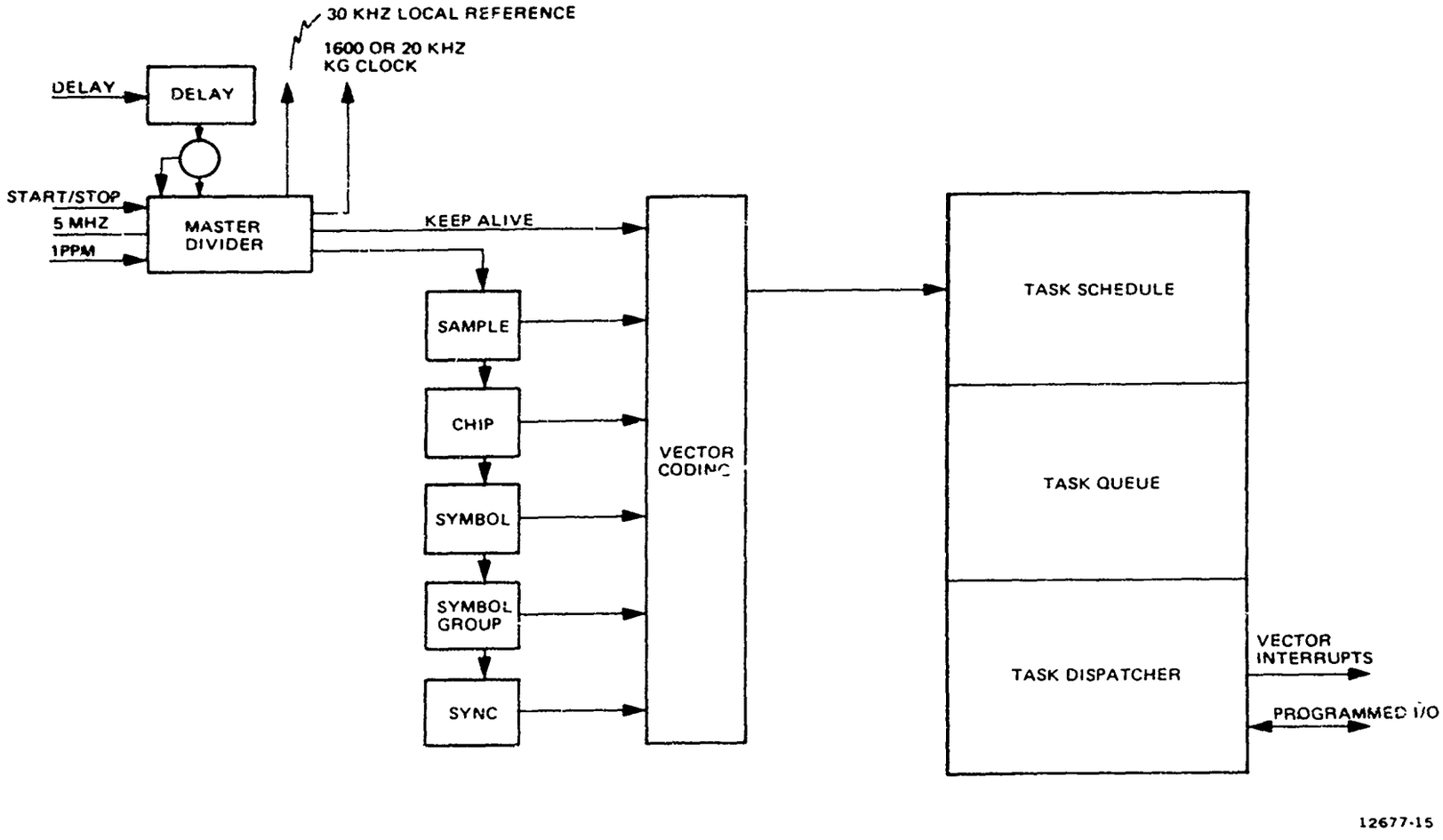


Figure I.1-1. Timing and Executive Module.

B. Vector Interrupts - Vector interrupts are sent to all modules within the unit to initiate processing and data transfer.

C. Programmed I/O - The programmed I/O is used to send buffer addresses, read status, or any other function which needs to be accomplished without direct module communication.

I.2 A/D MODULE

The function of this module is to mix quadrature demodulator timing with the received IF to extract the baseband signals containing the information. Figure i.2-1 is a block diagram of the IF mixing and sampling section. The incoming IF is mixed with the quadrature signals of the demodulator, filtered to limit the bandwidth of the signal plus noise, and sampled at the Nyquist rate (or higher). The sample data is then passed to the IF Pre-processing module via the DMA port to be further processed.

I.2.1 Signal and Timing Control

A.	IF –	This is a 500 mv RMS (nominal) signal from the receiver. IF center frequency is 7.5K. (Note - other characteristics such as impedence, etc., will not be included but can be found in the receiver manual.)
B.	*AGC –	This is a $\pm$5V signal slowly varying as a function of receive signal level.
C.	*Depth –	This is a 0 to +5V signal. Can be considered approaching DC in BW.
D.	Local Reference –	This is 4 times the IF frequency (30 kHz) and is divided into quadrature signals within the module. It is then used to extract baseband.
E.	Sample Time –	This signal initiates the A/D samples sequence. It occurs at least 4 times the baseband rate. We will use 6400 in this case.

I.2.2 Bus Data To Matched Filter Module

A. Sample Data - 9-bit, 2's compliment numbers sent to the IF Pre-processing module. They occur at the baud rate and are sent by DMA.

B. Status - Sent to the status panel at the baud rate.

I.3 IF PRE-PROCESSING MODULE

The function of this module is to reduce the signal sampling rate of the data received from the A/D Module to that of the matched filter input rate. The data is placed in an elastic buffer which is controlled by the sync module using fine sync adjustments to track doppler or to correct for sync mismatch.

*Not included in model.

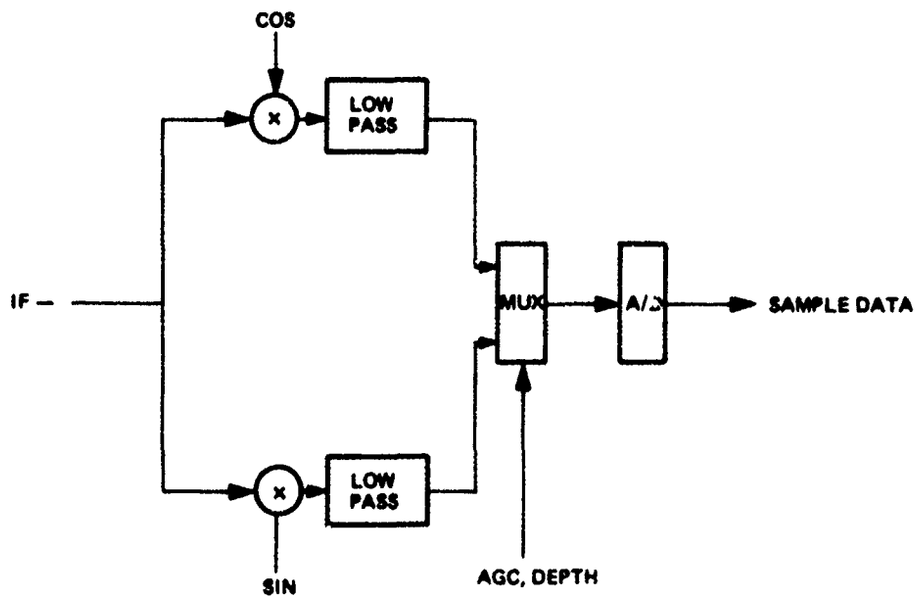


Figure I.2-1. A/D Module.

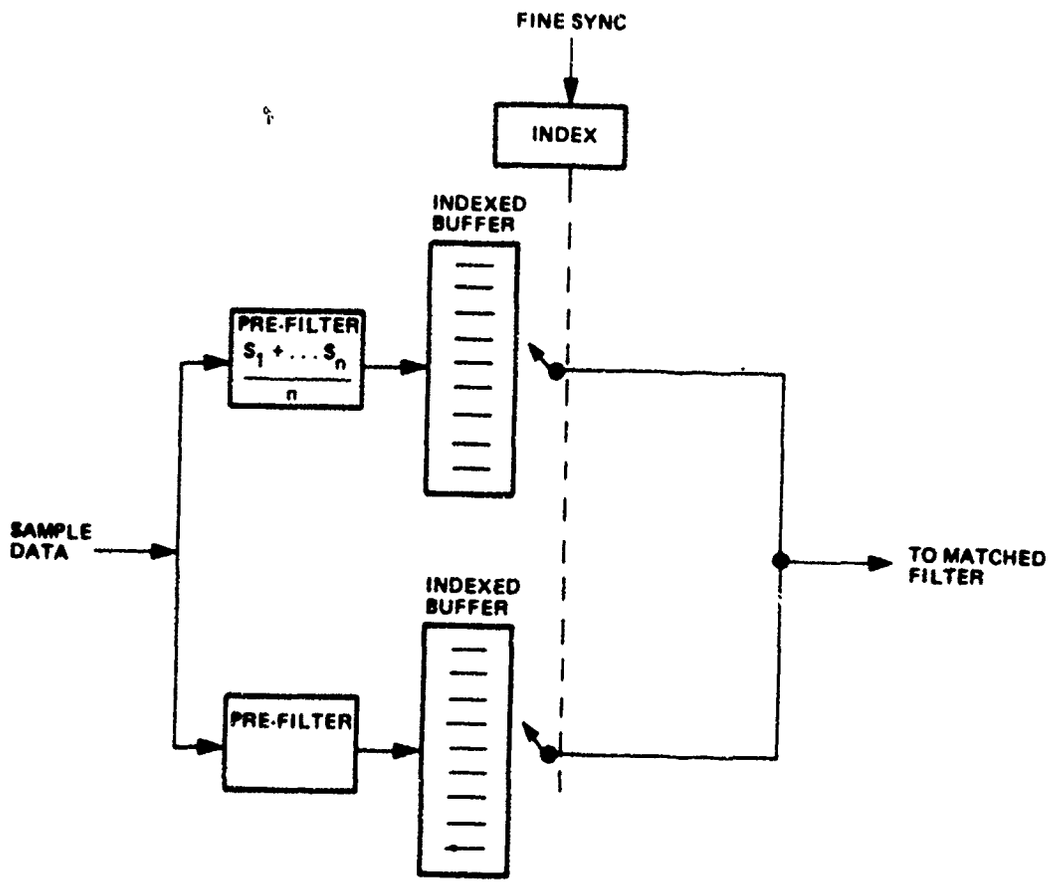


Figure I.3-1. Block Diagram of IF Pre-processing.

I.3.1 Inputs

A. A/D samples from the A/D Module. Received via the DMA channel at the sample rate. Two 9 bit 2's compliment number received at the 6400 Hz sample rate.
B. Fine sync-corrections received at the sync computation rate. Sent by the Sync Module via the DMA port.
C. Control - Two vector interrupts, one occurs at the A/D sample rate and one at the chip baud rate.

I.3.2 Outputs

A. Data is sent via the DMA channel to the Matched filter module at the baud rate (100 baud to 1600 baud). Two 16 bit 2's compliment numbers are transferred each baud period.
B. One status word is sent to the status panel at the baud rate.

I.4 MATCHED FILTER MODULE

This module receives sample data from the IF pre-processing module. The function of this module is to multiply the sample X and Y data with a digital 1/2 sine-wave representation in a transversal filter, sum the results and output the results to the next module which is the Correlation Module.

Figure I.4.1 is a block diagram of the Match Filter Module. In this implementation the sample data is separated into the X and Y substreams, prefiltered by averaging the samples, and generating subsample groups with eight samples used for each chip calculation. The indexed buffer shown in the block diagram is used for fine sync and doppler tracking. After the match filter operation occurs in the Transversal Filter, the chip commutation is applied giving the proper sine-cosine weighting to the received chip sequence. Data is then sent by the DMA port to the correlators.

I.4.1 DMA Ports

A. Sample Input Data
   Received via the DMA channel in a packed format. 4 samples of X and Y received each baud time. Nine bit 2's compliment notation representing $\pm$10 volts peak to peak. Baud time in this case occurs between 100 Hz and 1600 Hz rate.
B. Chip Output Data
   Output at the computation rate. Data is transferred via the DMA channel using the standard DMA controls.

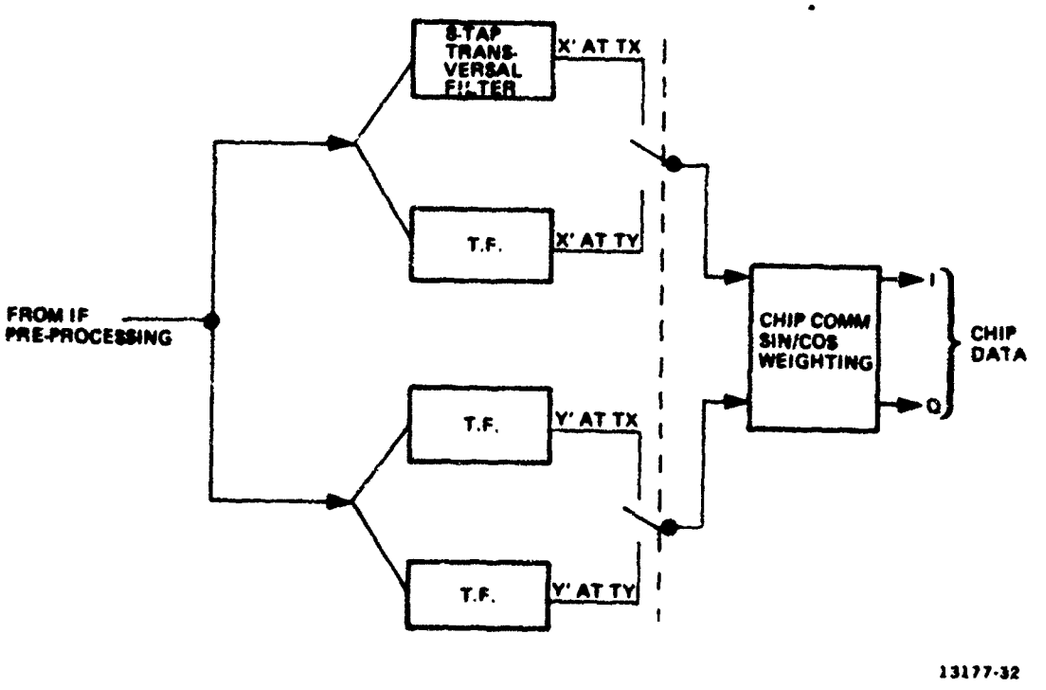


Figure I.4-1. Block Diagram of the Matched Filter Module.

I.4.2 Programmed Port

A. Fine Sync - This input is sent by the sync module and is used to move the fine sync pointer index, ± 1 from the current position. It is updated once each sync period.
B. Block Unload Address - This is a command word sent by the Exec specifying the address to transfer data into. This command is sent during initialization and is a 16 bit word.
C. Status - This is a sixteen bit word sent to the status panel at the chip rate to determine operational status of the moduie.

I.4.3 Timing

All timing is determined by the Executive. A vector interrupt is used to inform this module when to initiate an output transfer and calculate the new chips.

I.5 MULTI-WINDOW CORRELATION MODULE

The correlator receives chip samples from the Matched Filter Module, multiplies the sample with key, and sums the key/chip resultant over the defined symbol interval. The X and Y chip sums are delivered to the Symbol Processing Module at each symbol boundary time. Figure I.5-1 is a pictorial diagram of this operation. In this Signal Processing Module only one key per chip is used with a $\pm$ sum indicating a logic one or zero. The multiple windows shown are used to resolve time ambiguities in the signal propogation path.

I.5.1 Multi-Window Correlation Module Interfaces

A. DMA - Two DMA functions occur within this module. The first is the Input chip sample stream received from the Matched Filter module (See the MFM description for the data format, etc.). The second is the output correlation data stream. This output occurs at the symbol rate and is sent to the Symbol Processing module. The data is in the form of 16-bit 2's compliment numbers, blocked as shown.
B. Interrupt - Two interrupt Vectors are required by this module. The first occurs at the baud rate and initiates the multiply and sum operation. The second occurs at the symbol rate and initiates an unload and clear operation.
C. Programmed I/O
    1. Key Data - Read from the Key Demultiplex Module at the baud rate. One Key is required at each baud time. This is a 16 bit word of all 1's or 0's.
    2. Unload Address - This is an input from the Executive Module occurring during initialization specifying the 16 bit address to move symbol data to.
    3. Status - A 16 bit status word is sent to the status panel at the symbol rate.

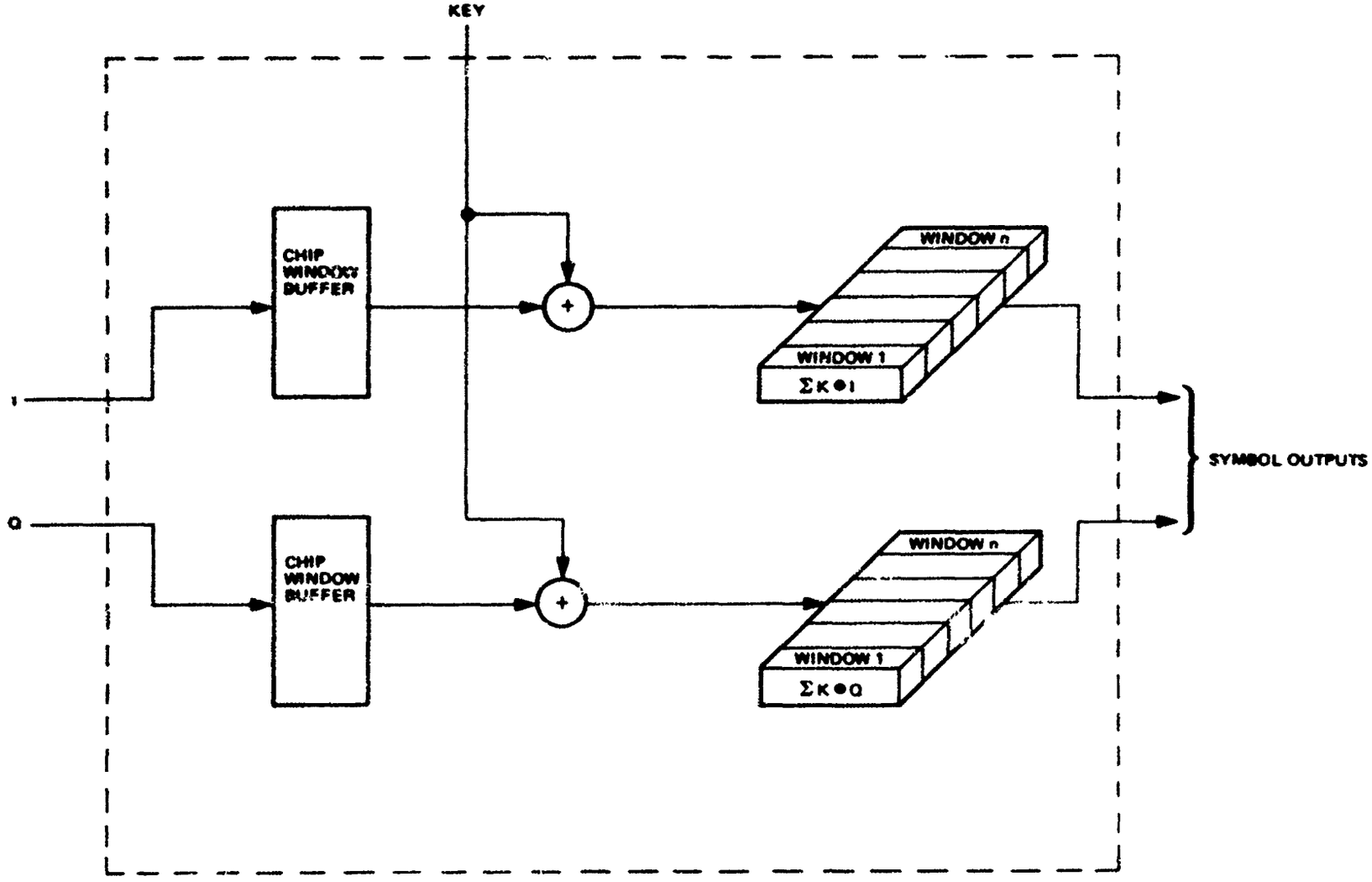


Figure I.5-1. Multi-window Correlation.

I.6 KG DEMUX MODULE

The functions of this module are to:

a) Provide KG control; i.e., start/stop high speed clocking, start lost speed clocking.
b) Real time input and run-up calculation
c) Input of key from key generator
d) Key demultiplex
e) KG failure monitor

This module interfaces directly with a Key Generator (refer to Figure I.6-1). It provides clock to the KG as received from the Timing module. The clock rate, high speed or low speed, is provided as designated by this module.

When an initialize request is received, real time is read from the cesium standard and giant step increments are calculated. A request for high speed clock is made to the timing module and fed to the KG. When run up is complete, low speed clock is requested and fed to the KG or the next minute pulse.

Raw key is input, demultiplexed and decrypted. This key is then stored as a -1 or a +0, and provided to the correlators.

The KG full operate line is also monitored and stored (-1 = KG Failure), and sent to the status panel at the baud rate.

Since the KG control module has access to time, a 24 hour clock (hours and minutes) will be kept and provided on request.

I.7 SYMBOL PROCESSING MODULE

The function of the symbol processing module is to reproduce the transmitted symbol from the correlated data inputs. This is accomplished by taking the dot product of the correlated data inputs as follows:

$$BIT = \left| 0^{o}_{c} \cdot 0^{o}_{p} + 90^{o}_{c} \cdot 90^{o}_{p} \right|_{T}$$

Where $0^{o}_{c}, 90^{o}_{c}$ = Correlated components of the current bit

$0^{o}_{p}, 90^{o}_{p}$ = Correlated components of the previous bit

T = Time uncertainties

Figure I.7-1 is a block diagram of the symbol processing module.

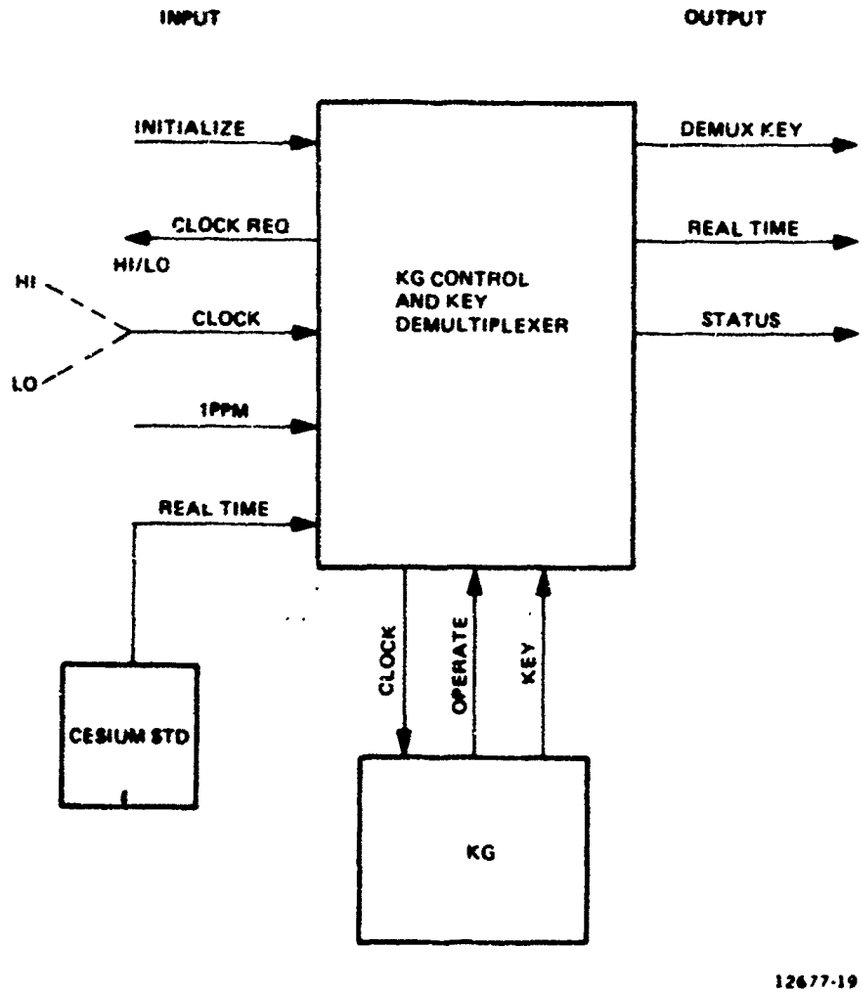


Figure I.6-1. Input/Output Lines.

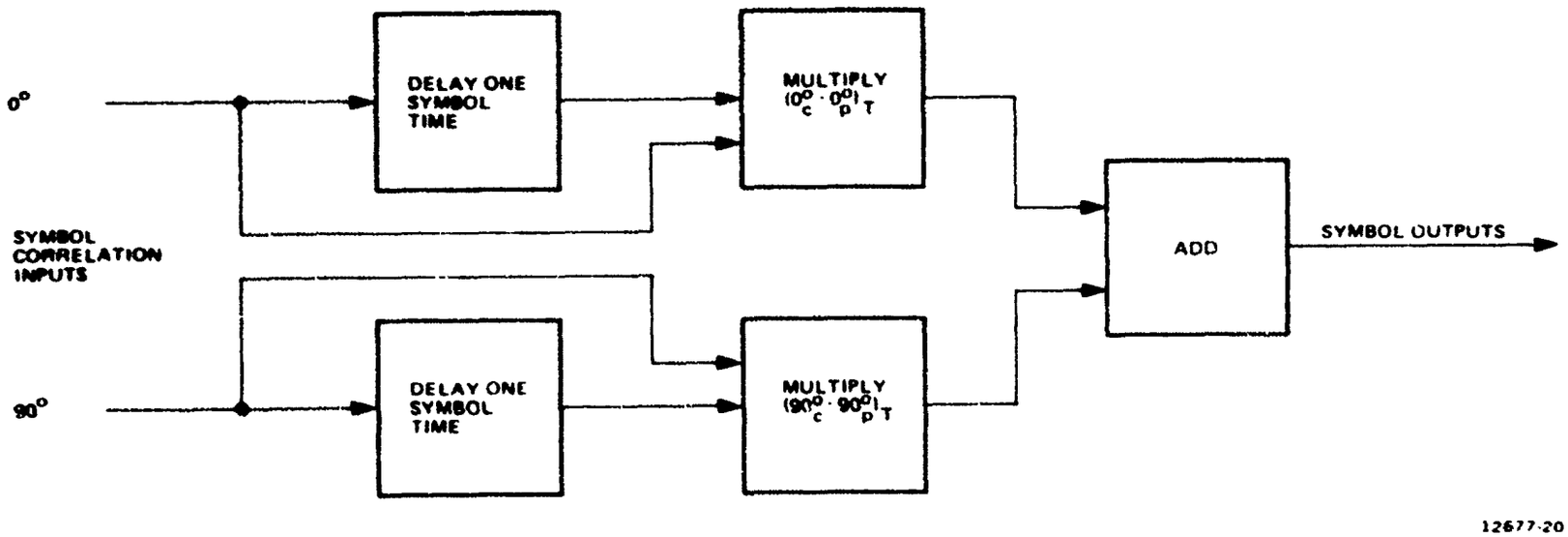


Figure 1.7-1. Symbol Processing.

I.7.1 Inputs

a) Input Control

Data is input into the module by DMA control. The DMA input was initiated by the Multi-Window Correlator Module. The interrupt shown is timed to occur after the DMA input is completed. The interrupt signals the module to begin its processing task.

b) Data Format

1) Eight 16 bit words of I (0^{O}) components maximum value = ($\pm$) 6350
2) Eight 16 bit words of Q (90^{O}) components maximum value = ($\pm$) 6350
3) One 16 bit word where bit 1 set = frame (phase) bit

c) Input Port

DMA

d) Multiplex Requirements

None

e) Timing

1) Rate: 17 words every bit time
2) Sequence: word 0 = frame indication
word 1-8 = I (0^{O}) component
word 9-16 = Q (90^{O}) component

f) Addressing

Single address

I.7.2 Output

a) Output Control

The module task was initiated upon a timed interrupt from the Timing and Executive Module. The data generated by the task is output twice by DMA control to first the Synchronization Module and then the Word Processing Module.

Also available for output is a 16 bit status word of the module condition. Output of this word is controlled by programmed I/O by the Timing and Executive Module.

b) Formats

1) Data
(a) Eight 16 bit words each consisting of values between -32767 to +32767.
(b) One 16 bit word where bit 1 set = frame (phase) bit.
2) Status - 16 bit word

c) Output Ports

Data: DMA

Status: Program I/O

d) Multiplex Requirements

None

e) Timing

1) Rate: Nine words (data) every bit time

One word (status) sent to the status panel every symbol time

2) Sequence: None

f) Addressing

Two addresses, serial data transmit

I.8 SYNC MODULE

The function of the Sync Module is to continually search for synchronization (presence of data transmission) over a pre-set time uncertainty and sync time period. Figure I.8-1 is a simple block diagram of the Sync Module. The data over which synchronization is determined is the bit data detected by the symbol processing module. The absolute values are accumulated over a predetermined sync period ('n' bit periods) across a preset window (time uncertainty). A sync threshold is calculated each sync time from the accumulated sync data. This threshold value is then used to determine if data is being transmitted and also its location within the time uncertainty. Fine sync calculation is made after sync has been detected and at the end of each block period thereafter until sync is lost.

I.8.1 Inputs

a) Input Control

Data is input into the module by DMA controlled input. The DMA is initiated and controlled by the symbol processing module. The interrupt (start process) shown is timed to occur after the DMA input is completed.

b) Data Format

1) Eight 16 bit words each consisting of values between -32767 to +32767.

2) One 16 bit word where bit 1 set = frame (phase) bit.

c) Input Port

DMA

d) Multiplex Requirements

None

e) Timing

1) Rate: 9 words every bit time

2) Sequence: Synchronize to a frame (phase) bit

f) Addressing

Simple address

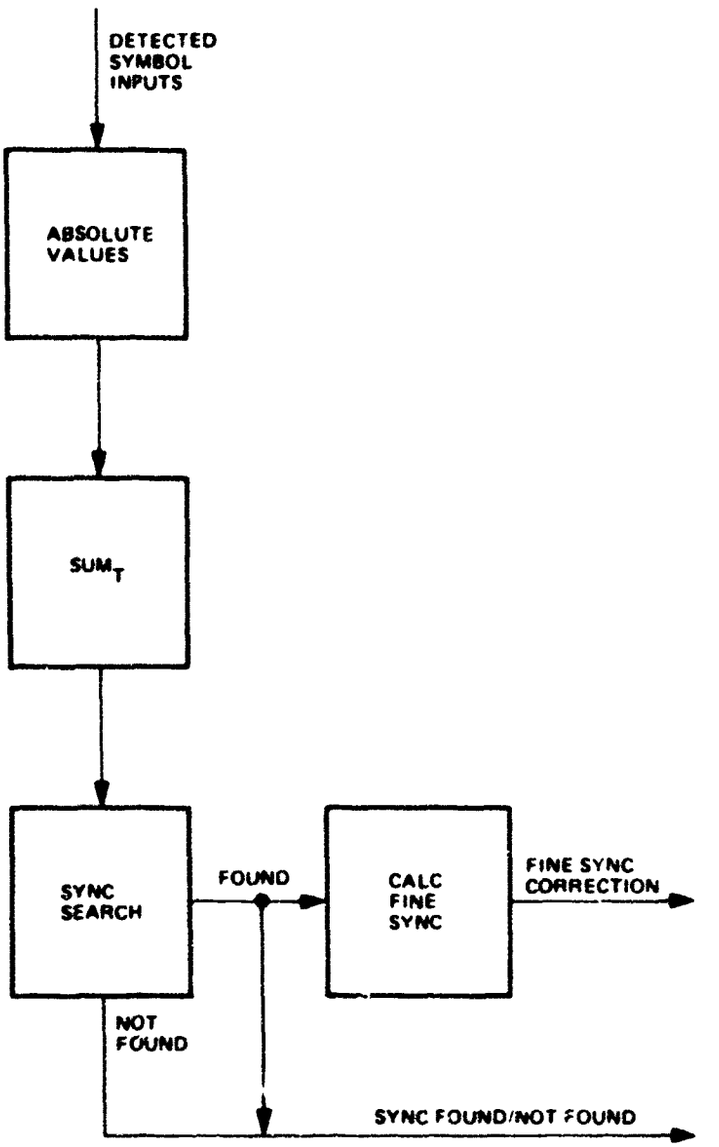


Figure I.8-1. Sync Module.

I.8.2 Output

a) Output Control

There are three words generated for output by the module:

1) Sync/No Sync Indication with Sync Location
2) Fine Synchronization Correction Direction
3) Module Operation Status

These output words are accessed (read) by the Executive Module via programmed I/O. The read intervals are controlled (scheduled) by the Executive Module, and also provided to the appropriate modules at the correct time intervals.

b) Data Format

1) To Word Processing: One 16 bit word identifying sync found and the location or sync not found.
2) To Chip Processing: One 16 bit word with fine sync correction value of +1 or -1.
3) Status: One 16 bit word

c) Output Port

Program Control I/O

d) Multiplex Control

None

e) Timing

1) Rate: Sync Found/Lost - 1 per sync period
   Fine Sync Correction - 1 per block
2) Sequence: None

f) Addresses

Three - Sync Word
Correction Word
Status Word

I.9 WORD PROCESSING MODULE

The function of the word processing module is to reconstruct the code word, detect and correct any errors by forced erasure schemes, and finally to extract the characters for output.

Figure I.9-1 is a simple block of the Word Processing Module. Identical to the Sync Module, the word processing module receives the bit data detected and processed by the symbol processing module. The maximum data stored is equivalent to the sync period and is discarded if sync is not obtained.

However, if sync is detected, the code word is constructed using the sync found index to extract the bit data from the bit buffers. A unique parity check is made and if bad, a forced erasure correction is made

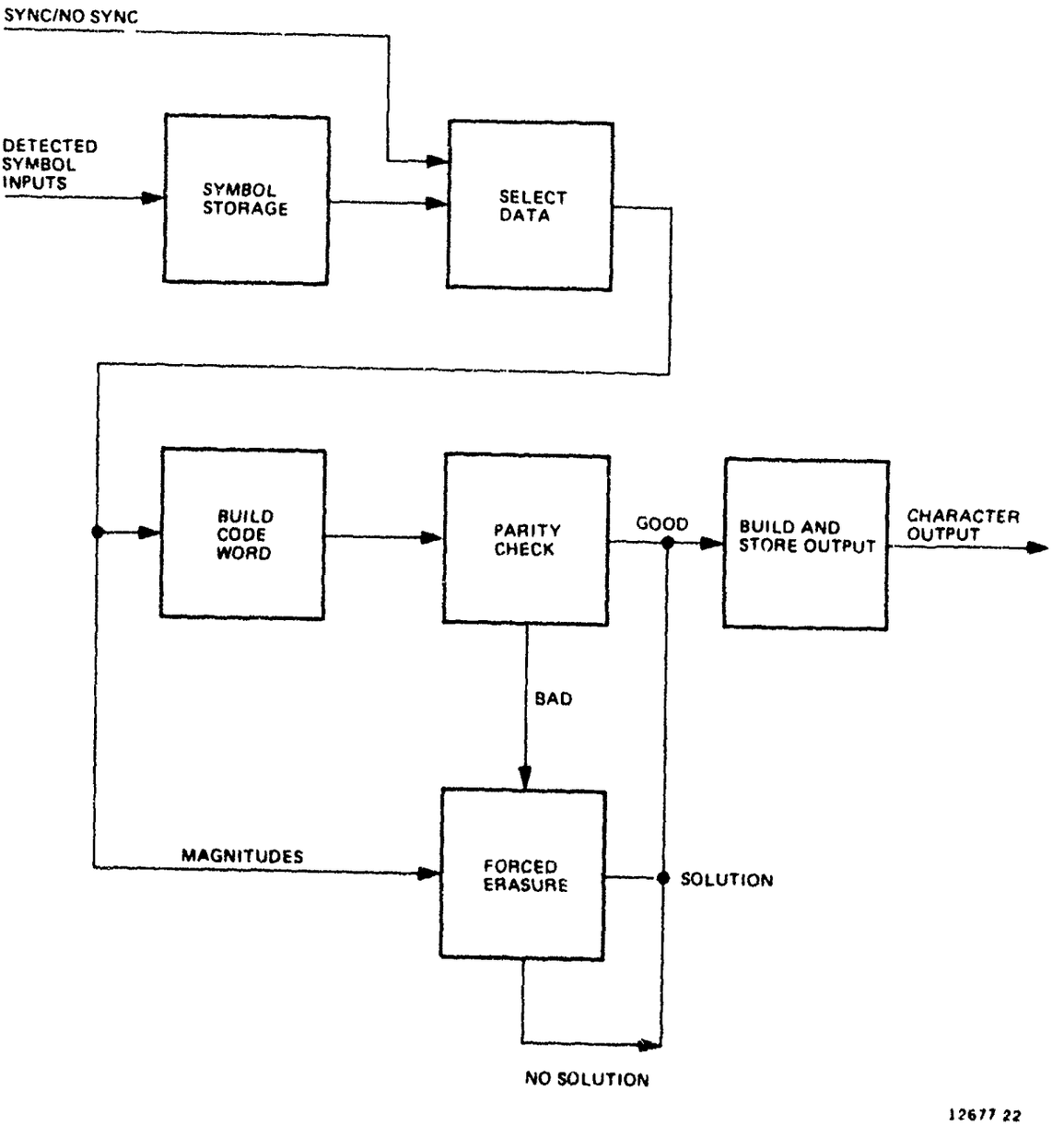


Figure I.9-1. Word Processing Module.

using the associated bit magnitudes. The characters are extracted from the code word and stored in the output buffer for output to the device. Appended to the characters is one of the following generated message (optional):

a) No Error Detected
b) Errors Detected and Corrected
c) Errors Detected and Uncorrectable

I.9.1 Input

a) Input Control
   Data is input into the module by DMA control, initiated and controlled by the symbol processing module. The interrupt (start process) shown is timed to occur after the DMA input is completed. The sync data word is input just prior to the interrupt by the Executive Module by programmed I/O. The module begins processing on receipt of the interrupt and sync notification.
b) Data Format
   1) Bit Data
      (a) Eight 16 bit words each consisting of values between -32767 to +32767
      (b) One 16 bit word where bit 1 set = frame (phase) bit
   2) Sync Data
      One 16 bit word identifying sync found and the location, or sync not found/lost.
c) Input Port
   1) Bit Data: DMA
   2) Sync Data: Program Control I/O
d) Multiplex Control: None
e) Timing
   1) Rate: Bit Data - 9 words each bit time
      Sync Data - 1 word each sync time
   2) Sequence: Synchronize bit data to frame (phase) bit
f) Addressing
   Two Addresses

I.9.2 Output

a) Output Control
   'n' number of characters are buffered by the word processing module, and are output by DMA control to the output device when signaled by a 'data output' interrupt. Module status is also available for output. This status word is output via programmed I/O by the Executive Module.
b) Formats
   1) Data: 'n' number of 16 bit words, three 5 bit baudot characters per word.
   2) Status: 16 bit word

c) Output Port:
   1) Data: DMA
   2) Status: Programmed I/O

d) Multiplex Control
   None

e) Timing
   None

f) Addressing
   Single address

I.10 STATUS PANEL MODULE

This module acts as a slave to the other modules and receives a status word from each, updated at the individual module rates. In the general case this could also be a control panel which disable timing interrupts during the module initialization phase or could be scanned by the executive and control module at some pre-determined rate.

APPENDIX J

FUNCTIONAL MODULES OF A MESSAGE SWITCH

FUNCTIONAL MODULES OF A MESSAGE SWITCH

The identification of modular elements within a message switching system was performed as a preparatory step to the analysis of alternative modular system designs. The hardware or software realization of these modules and their roles in a total system are described below. Functional specifications with comments relative to application to a small switch follow this introductory material.

A message switching system can be divided up into various functional modules. One such list is as follows:

A. Line Interface
B. Input Service
C. Output Service
D. Message Storage
E. Message Journaling
F. Validation
G. Routing
H. System Control
I. Memory Management
J. Error/Recovery
K. Initialization
L. Diagnostic/Test

Any of the activities of a message switch can be found under one or more of these functions. Also there will be a hierarchical relationship that exists among these functions. Exactly how each function fits in the system hierarchy will depend to a large extent on the switch's implementation. Because this relationship exists, it may require different techniques to fully describe each function.

To describe the previously listed functions, a module definition specification was devised. The specification requires each module to be defined in terms of its input, output, and transfer operations. For some functions these requirements provide an easy technique for describing the function performed. For example the input/output service modules are easily described in this format. Using this specification on functions such as System Control proved to be very difficult. This difficulty arises from the fact that System Control is really a group of independent activities that in combination provide control of the system. Thus, the modular description serves as a means of identification and assures consideration of each function during the design process. The implementation of System Control, however, will be as part of a system executive function.

By dividing the list of functions into two inherently different classes, each class can be described in a manner suitable to it. The first class will contain those functions that are assigned primarily to switching

messages. The second class will contain those secondary functions that allow the primary functions to communicate with each other. Thus, the secondary functions are relatively independent of "one for one" correspondence with the number of system lines. A close correspondence of this type does exist for the primary functions.

The primary functions will be hierarchically lower than the secondary ones. Dividing the initial list, the following primary functions, with the steps they perform, and secondary functions are obtained: (Refer to Figures J-1 and J-2).

PRIMARY MODULES

I. Line Interface

A. Input
   1. Receive modulated signal
   2. Clock recovery
   3. Demodulate signal to NRZ
   4. Synchronize bits
   5. Synchronize characters
   6. Generate interrupt to Input service

B. Output
   1. Modulate NRZ signal
   2. Outputs modulated signal using line driver

II. Input Service

A. Fetches character from Line Interface
B. Checks protocol
C. Translates characters (if needed)
D. Adds header
E. Reformats (if needed)
F. Stores message in main memory using address from mailbox
G. Signals complete message reception in mailbox
H. Initial logging; write header to log if non-perishable

III. Edit/Validation

A. Fetches message header from main memory
B. Verifies information in each field
C. Crosscheck information between fields
D. Replaces header in main memory if modified
E. Signals results of edit/validation

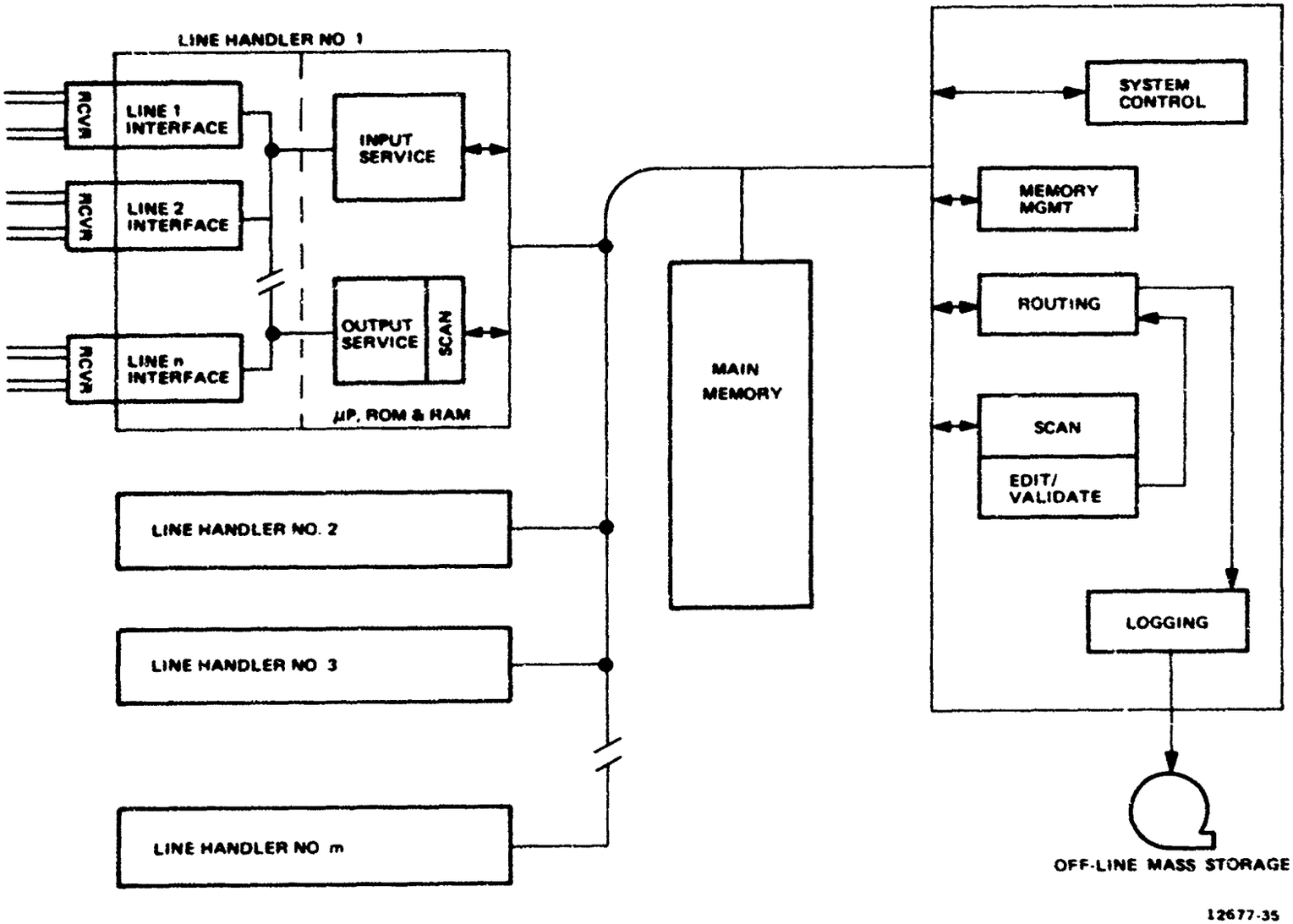
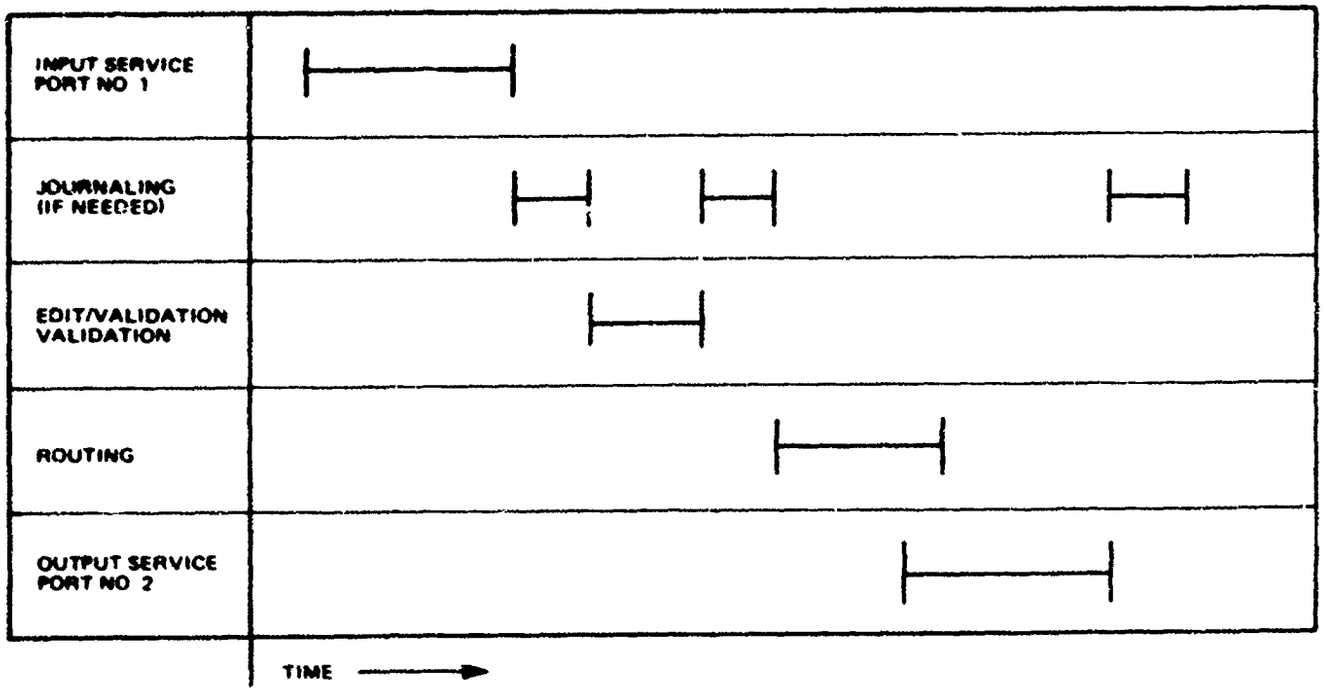


Figure J-1. "Strawman" Message Switch.



STEPS NEEDED TO SWITCH A MESSAGE FROM PORT NO 1 TO PORT NO 2 WITH JOURNALING
THIS CHART ASSUMES NO OTHER MESSAGES ARE ACTIVE IN THE SWITCH AT THIS TIME AND
THAT OVERHEAD FUNCTION TIME IS NEGLIGIBLE

12677 36

Figure J-2. Timing Chart.

IV. Routing

A. Fetches output destinations from message header
B. Determine output service module and line addresses for each destination and construct route control words (RCW) for each
C Supervises output service mailbox protocol
D. Write to log indicating disposition of message copy; multiple entries for multiple addresses

V Output Service

A. Polls its mailbox for new RCW
B. Fetches message from main memory
C. Translates message (if needed)
D. Modifies header (if needed)
E. Adds line protocol
F. Transfers characters to Line Interface for output

SECONDARY MODULES

I. INITIALIZATION
II. MESSAGE STORAGE
III. MEMORY MANAGEMENT
IV. SYSTEM CONTROL
V. ERROR/RECOVERY
VI. DIAGNOSTIC/TEST

The following is a description of each primary module in the form Input-Output-Transfer. Only a general description of the secondary modules (except the Message Storage Module) has been included.

SWITCH FUNCTION MODULE: LINE INTERFACE MODULE, TYPE: PRIMARY

I. General

The Line Interface Module provides the necessary hardware interface for the line handler portion of the message switch with the subscriber lines or loops. It performs such functions as modulation and demodulation of signal to and from NRZ signals, clock recovery and bit synchronization, line driver and receiver terminations, and the insertion of framing signals or characters as required by the subscriber. Multiple line interface modules may be contained within a single Line Handler, depending on the speed of the subscribers being serviced. A Line Handler contains a single processor for servicing the I O's.

A. Line Interface

Input Data Line (Input)

The input from the subscriber may be either two-wire balanced or unbalanced lines. The lines may contain some modulated signal which must be demodulated to NRZ or they may contain NRZ signals. Such modulated signals as conditioned di-phase and others allow the line interface module to derive clock; whereas, an NRZ signal must be accompanied by a clock input since clock cannot be derived from it. The signal is taken off the line via an appropriate signal receiver.

Output Data Line (Output)

The output to the subscriber, like the input from the subscriber, may be NRZ or a modulated signal. If the line contains synchronous data, the line interface may generate a sync pattern or character as required by the subscriber. The signal is applied to the line via the line driver.

Timing In (Input)

Message Switch line handler timing is supplied by the line device via this line in the event that NRZ signals or other signals are on the input lines that cannot be used for clock recovery. This assumes that the line device supplies master timing.

Timing Out (Output)

If the subscriber derives its clock from the Message Switch and the subscriber cannot derive the clock from the type signal on the output line, this line is used by the Message Switch to supply the timing.

B. Processor Interface

Data Bus (Input/Output)

This bus is an 8-bit, parallel, bidirectional bus used for transfer of data to and from the line handler processor. When data is to be passed to the processor, the data is held in an 8-bit data register until the processor accepts the vectored interrupt from the line interface unit. The processor then addresses the line interface unit and "Fetches" the character, using the I/O Read line.

Address Bus (Input/Output)

This is a 16-bit bus which is used by the line handler processor to address the various line interface modules and the memory locations. Only 5-bits of this bus are required for line interface modules since the number of these modules will probably never exceed 32 for a single line handler. In order for a line interface module to pass or receive data to or from the processors, it must recognize its address on the address bus.

I/O Read (Input)

This line is supplied to the line interface module, along with the address, by the line handler processor to load the data word from the data bus into the module.

I/O Write (Input)

This line is supplied to the line interface module, along with the address, by the line handler processor to load the data word from the data bus into the module.

Input Word Ready (Output)

This line serves as the interrupt to the Line Handler Processor, informing it that a character has been received, and is ready to be fetched and processed. The address bus will contain the address of that Line Interface to Input Service Module interrupt routine.

Ready For Next Word (Output)

This is an interrupt to the Line Handler Processor, addressing the line interface to Output Service Module interrupt routine. This routine will either output another character or set an output status flag, and reset the interrupt.

Loss of Bit Sync (Output)

This line is a status report to the line handler that bit synchronization has been lost. What is done about the report is a system definition.

Clock Input (Input)

This line contains the line handler clock and is used by the synchronization circuits to generate clock.

II. Inputs

A. Input Data Lines

1. Format:

   a. Transmission Inputs

      Signal may be conditioned-diphase or NRZ serial. Code may be 7-bit ASCII or 5-bit Baudot asynchronous with a start bit, a parity bit, and 1 or 2 stop bits. Code may also be 7-bit ASCII synchronous and 1 bit of parity.

   b. Processor Input

      5-bit Baudot or 7-bit ASCII characters.

2. Input Ports:

   The Line Interface Module is a bilateral device. It receives data from the transmission lines and also from the processor data bus.

   a. Transmission Line Input Port

      This input port is a receiver that is designed for detecting the type signal on the line; i.e. MIL-STD-188C, MIL-STD-188-100, etc.

   b. Processor Data Bus

      This input is an 8-bit parallel bus (bidirectional).

3. Multiplex Requirements:
   None.

4. Timing (Rate):
   a. Transmission Line Input Port
      Inputs vary from 45.5 bits/sec Baudot asynchronous to 16,000 bits/sec ASCII synchronous.
   b. Processor Data Bus
      Inputs are 8 parallel data lines, 16 paralleled address lines and 6 parallel control lines.

5. Timing (Sequence):
   a. Transmission Line Input Port
      Character bits are input as they are detected, in the asynchronous system. In the synchronous system, characters are received in time with the generated clock.
   b. Processor Data Input
      Characters are input in time with address and control lines, asynchronously

6. Addressing:
   a. Transmission Line Input Port
      No addressing
   b. Processor Data Input Port
      One address per character received.

B. Input Control Lines
   1. Format:
      a. Timing In
         This is a standard square wave pulse chain running at the rate of the data being received ranging from 45.5 pulses/sec to 16 K pulses sec. Used only when data is NRZ.
      b. Clock Input
         This is a standard square wave pulse chain running at some multiple of the data being received on the Transmission Line Input.
      c. I/O Read/I/O Write Inputs
         The lines are levels only and are activated by the processor when the Line Interface Module is to read or write, respectively.

   2. Input Ports:
      a. Timing In
         A receiver designed to detect the signal on the line; i.e., MIL-STD-188C, MIL-STD-188-100, etc.
      b. Clock Input
         A standard pin in.

c I/O Read/I/O Write Inputs

Standard pin ins, a part of the processor control bus

3 Multiplex Requirements

None

4 Timing (Rate)

a Timing In

This input only exists if NRZ is being received on the Transmission Line Input (See para II B 1.a)

b Clock Input

(See para. II.B 1.b)

c I O Read I O Write

Asynchronous (See para II.B.1.c)

5. Timing (Sequence):

a Timing In

(See para. II.B.1.a)

b Clock Input

(See para. II.B.1.b)

c. I/O Read/I/O Write

(See para. II.B.1.c)

6. Addressing

Not applicable

III. Outputs

A. Output Data Lines

1. Format:

a. Transmission Outputs

Conditioned diphase or NRZ serial data. Code is either 5-bit Baudot or 7 bit ASCII asynchronous, or 7-bit ASCII synchronous with appropriate start, stop, and parity bits

b. Processor Outputs

5-bit Baudot or 7-bit ASCII

2. Output Ports:
   a. Transmission Output Ports
      An appropriate driver for producing required waveform on line; i.e., MIL-STD-188C driver, MIL-STD-188-100 driver.
   b. Processor Output Ports
      8-bit Parallel bus (bidirectional)

3. Multiplex Requirements:
   None.

4. Timing (Rate):
   a. Transmission Output
      (See para. II.A.4.a)
   b. Processor Output
      (See para. II.A.4.b)

5. Timing (Sequence):
   a. Transmission Output
      Character bits are output at the required line data rate varying from 45.5 bits/sec to 16 K bits/sec.
   b. Processor Data Output
      Characters are output in time with address and control lines, asynchronously.

6. Addressing:
   a. Transmission Output
      Not applicable.
   b. Processor Data Output
      One address per character output.

B. Output Control Lines
   1. Format:
      a. Timing Out
         This output exists only if the message switch is the master time source. It is a symmetrical square wave pulse chain running at a rate depending on the data rate of the transmission line and varies from 45.5 pulses/sec to 16 K pulses/sec.
      b. Loss of Bit Sync
         This output is a DC level.
      c. Input Word Ready
         This output is a DC level.
      d. Ready for Next Word
         This output is a DC level.

2. Output Port
   a. Timing Out
      An appropriate driver for producing required waveform on line; i.e., MIL-STD-188C, MIL-STD-188-100, etc.
   b. Loss of Bit Sync
      Standard pin out to processor.
   c. Input Word Ready
      Standard pin out to processor.
   d. Ready for Next Word
      Standard pin out to processor.

3. Multiplex Requirements:
   None.

4. Timing (Rate):
   a. Timing Out
      (See para. III.B.1.a)
   b. Loss of Bit Sync
      Raised when bit sync is lost. Raised in sync with Clock Input signal.
   c. Input Word Ready
      Raised when Line Interface Module has word to send to processor. Asynchronous.
   d. Ready for Next Word
      Raised when Line Interface Module is ready for another word to put on the transmission line. Asynchronous.

5. Addressing:
   None.

IV. Transfer Characteristics

A. Transmission Input Data to Processor Data Bus

1. Throughput Delay (Data Rate = 16 kb/s).
   Time to receive 8-bit serial character at 16 K B/S is 500 μsec. This is a function of the data rate.
   Time to convert from serial to parallel is 62.5 μsec. This is assuming internal clock running at data rate, 16 K B/S.
   Time to transfer character from Line Interface Module to Input Service function is 10 μsec.
   Total throughput delay per character is 572.5 μsec.

B. Processor Data Bus to Transmission Output

1. Throughput Delay (Data Rate = 16 kb/s).

   Time to receive 8-bit character (parallel) from Input service function is approximately 10 μsec.
   Time to convert from parallel to serial and transmit out at 16 K B/S is approximately 562.5 μsec.
   Total throughput delay per character is 572.5 μsec.

SWITCHING FUNCTION MODULE: INPUT SERVICE MODULE, TYPE: PRIMARY

I General

The Input Service Module is a software module, resident in the Line Handler Processor, that directs the checking of receive protocol, the translating of codes, the checking of parity, and the reblocking of messages. It also passes the message blocks to common memory for further processing and routing. The module is called by the Line Interface to Input Service Module interrupt routine. This module is one of several that is implemented within a Line Handler Processor.

A. Subscriber Protocol

The Input Service Module performs all protocol checks on messages received by the switch. After the message has been demodulated to NRZ and formatted into 8-bit bytes by the Line Interface Module, it is "fetched" by the Line Handler Processor, under direction of the Input Service Module, and checked for its protocol legitimacy; i.e., SOM, STX, EOM, etc. If protocol is correct, the parity of each message byte is checked and if block parity check is required, it is performed at the end of the message block. If the message is being received from an ASCII type subscriber, it is stored byte-by-byte into the Line Handler local storage for further processing.

B. Code Translation

If the message is being received from a subscriber that uses Baudot code, it must be translated to ASCII, which is the code used by the switch. The bytes are translated by the code translation routine and placed into local memory for further processing.

C. Message Reformatting

If relocation of bytes within the message header are required, the Input Service Module controls the Line Handler Processor in performing this task. Also, if messages are segmented for transmission, this module performs this task, adding the date/time group and the originator routing indicator to the header, as required.

D. Message Transfer

The Input Service Module controls the Line Handler Processor in transferring the message to common memory for further processing and routing by the central processor. The Line Handler Processor, under control of the Input Service Module, reads from a "mail box" in common memory the address

at which the message from the Line Handler should begin. The Line Handler transfers the message, one character at a time to common memory in segments made up of multiple bytes. The segments are chained in common memory by the Line Handler to form the complete message.

II. Inputs

A. Format: Character oriented messages; each character is 7-bit ASCII with 1 parity bit, or Baudot with parity bit.

B. Input Ports: (1) 8-bit parallel data; (2) 16-bit parallel address; (3) 4-bit parallel control lines; (4) "fetch" from Line Interface Module.

C. Multiplex Requirements: None. The system bus is dedicated to this task until complete.

D. Timing (Rate): The timing varies with application. It will vary from a low of 5.7 char/sec asynchronous Baudot to a high of 1200 char/sec synchronous ASCII. All transfers are one character at a time.

1. Asynchronous Baudot. Since all transfers occur on a 8-bit byte basis, a Baudot character, which is six-bits long (including parity) is transferred with two ignored bits. Because "letter shift" and "number shift" characters are required to obtain the complete character set in Baudot, it is assumed that for every 11 Baudot characters received 10 ASCII characters are output.

2. Synchronous ASCII. Since no translation occurs the output equals the input.

3. Message Content. A date/time group is added to the header before characters are output, thus the output message contains more characters than the input message. Assume 8 characters added for every 160 characters; i.e., for 160 chars-in there are 168 chars-out (1:1.05 ratio).

E. Timing (Sequence): Transfer of data is performed on data lines under control of the processor control lines, asynchronously.

F. Addressing: Two addresses per character transferred are required - one for "fetching" from Line Interface Module buffer and one for loading to local memory.

III. Outputs

A. Format: Character oriented messages; 8-bit ASCII characters.

B. Output Port: 8-bit parallel data, 16-bit parallel address, 4-bit parallel control lines; load to common memory.

C. Multiplex Requirements: None.

D. Timing (rate): See Inputs.

E. Timing (sequence): See Inputs.

F. Addressing: Two addresses per character transferred are required - one for "fetching" the character from the local memory and the other for loading the character to common memory.

G. Other Outputs: Defined under Output Port.

IV. Transfer Characteristics

A. Throughput Delay. Each input word produces an equivalent output word. Delay between input word and output word as a result of storage of message, translation of codes (if required), reformatting of header, and loading to Common Memory is approximately 2 ms per character.*

SWITCHING FUNCTION MODULE: OUTPUT SERVICE MODULE, TYPE: PRIMARY

I. General

The Output Service Module, a software subroutine resident in the Line Handler Processor, polls the mailboxes in the common memory, which contain pointers to output messages which are to go to subscribers via the Line Interface Module. This module also controls the output protocol with the subscriber, and translates the message into Baudot, if necessary. Output Service also controls the building of message headers, and the reblocking of messages, as required.

A. Memory Transfer

The Output Service Module causes the Line Handler Processor to poll specified locations in common memory to determine if messages have been queued for any of the subscribers being serviced by this particular Line Handler. When a message is ready, it is transferred by the processor, byte-for-byte, to the Line Handler local storage, going through Code Translation, if necessary.

B. Code Translation

If the message is being sent to a subscriber that uses a different code than the message switch, the bytes are translated by the code translation routine and placed back in local memory for further processing. This selection is fixed at design time.

C. Message Reformatting

If the subscriber requires a unique type header for the message, that header is constructed by this routine. See Basic Message Layout (Figure F-2) for details on the header.

D. Output Protocol

The Output Service Module controls the Line Handler Processor in generating the proper protocol for the subscriber; i.e., SOM, STX, EOM, etc. This module also controls the transfer of the message

*The ULMS application is characterized by a maximum arrival rate of 1 char/3.5 ms.

bytes into the Line Interface Module for transmission. The module recognizes ACK or NAK from the subscriber and resends a message, if required, after the receipt of a NAK. The module also adds character parity and block parity, if required by the subscriber.

II. Inputs

A. Format: Character oriented messages; 8-bit ASCII characters.

B. Input Port: 8-bit parallel data; 16-bit parallel address, 4-bit parallel control lines; "fetch" from common memory.

C. Multiplex Requirements: None.

D. Timing (Rate): Timing varies with application. It will vary from a low of 5.7 char/sec asynchronous to a high of 1200 char/sec. In a message processing mode, the Output Service Module transfers 80 characters per transaction (a block); in a scanning mode, the Output Service Module transfers two characters per transaction (mailbox contents). All transfers from common memory to Line Handler Storage are one character at a time, asynchronous. If the characters are translated to Baudot additional characters will be added to account for shift up (alpha characters) and shift down (numeric characters). Assume an average of an additional shift up or shift down character for every ten characters of text. Therefore for each 10 characters in, 11 characters are output.

E. Timing (Sequence): Transfer of data is performed asynchronously upon command of the processor via control lines.

F. Addressing: Two addresses per character transferred are required - one for "fetching" from common memory and one for loading to local memory.

III. Outputs

A. Format: Character oriented messages; each character is 7-bit ASCII with 1-bit parity or baudot with parity.

B. Output Port: 8-bit parallel data, 16-bit parallel address, 4-bit parallel control lines; load to Line Interface Module.

C. Multiplex Requirements: None.

D. Timing Rate: See Inputs.

E. Timing Sequence: See Inputs.

F. Addressing: Two addresses per character transfer - one address for "fetching " from local memory, one address for loading to Line Interface Module.

G. Other Outputs: Defined under Output Port.

IV. Transfer Characteristics

Throughput Delay: Each input word produces an equivalent output word with an additional output word approximately every 10 input words. Delay through the module as a result of fetching each character from common memory, translating it, reformatting the header, placing it in local storage, and then transferring it to the Line Interface Module is approximately 2.5 ms.* However, the character will be held in local storage (RAM) until requested by Line Interface Module.

SWITCHING FUNCTION MODULE: MESSAGE JOURNALING MODULE, TYPE: PRIMARY

I. General Description

This module maintains historical records of all traffic which has been processed by the switch. These records are designed for long-term or permanent retention and infrequent recall. The Message Journaling Module is logically and electrically independent of the Message Storage Module.

Normally, all processed messages in the switch will be reflected in an entry in the Message Journaling Module. As a minimum, this entry will include:

A. Path Field (indication of message routing prior to arrival at switch);
B. Header (including Precedence, Classification, Sender and Addressee Routing Indicators, Message Type, and Message Text Code and Format);
C. Date/Time of Message receipt; and
D. Date/Time and manner of disposition of each message copy.

A non-perishable message may be preserved in its entirety, rather than simply having its header recorded. Items (A) through (C) above may be received from the Message Storage Module prior to processing by the Validation and Routing Modules, and Item (D) from the System Control Module after final disposition.

II. Inputs

A. Format: Character Strings (messages or message headers), 7-bit ASCII with one parity bit per character; string length 25-200 characters (message headers) or 500-8000 characters (messages).

B. Input Port: DMA.

---

*The maximum rate out for the application selected (ULMS) is 4.5 ms per character.

C. Multiplex Requirements: None.

D. Timing (rate): Must be able to accept characters at a rate fast enough to store the data listed in section I, plus the current processing status (requires three inputs per message), without building up a queue.

E. Timing (sequence): Address and enable signals required no later than data characters; if internal address translation is done for each address (e.g., on disk) then address signal required same period prior to data character; actual timing depends on component/subsystem choices.

F. Addressing: One address per data character; address length dependent on journal size and mapping techniques used.

III. Outputs

A. Format: Same as Input.
B. Output Port: DMA.
C. Multiplex Requirements: None.
D. Timing (rate): Output only on manual command; 10 or more input words per output word.
E. Timing (sequence): Same as Input.
F. Addressing: Same as Input.

IV. Transfer Characteristics

A. The Message Journaling Module will store the information need for each message (see Section I) but will normally only output when a message is not delivered.

SWITCHING FUNCTION MODULE: VALIDATION MODULE, TYPE: PRIMARY

I. General Description

Message validation functions divide easily into two general categories:

A. Format Validation: Insuring that incoming messages contain a certain minimum amount of information, and contain it in certain locations in the message. Example: An incoming message must contain a SOM (Start of Message) sequence, a STX (Start of Text) or EOH (End of Header) sequence, and an EOM (End of Message) sequence.* This delimits and segregates a message from the balance of a (potentially continuous) input bit stream. Format validation is performed by the Input Service/Code Translation Module.

---

*Depending on the message formats, there may be a minimum and maximum number of message characters between SOM and STX/EOH, and between STX/EOH and EOM.

B. Content Validation: Insuring that the contents of the message headers are such that delivery of the message is allowed under network rules. For example:

- Precedence - is the originator authorized to transmit messages with the stated precedence?
- Classification - is the originator authorized to transmit messages with the stated security classification? Are the addresses authorized to receive messages with the stated security classification?
- Message Type - does the message type indicator correspond to an allowable message type (perishable/nonperishable)?
- Message Text Code and Format - do these indicators correspond to allowable text codes and formats?
- Addresses - do the originator and destination routing indicators (addresses) refer to valid subscriber locations within the network?

This content validation implicitly validates the internal format of the header (by assuming that precedence, classification, and other indicators are located at specific points within the header). Content validation is performed by the Validation Module in the CPU.

The Validation Module receives message headers from the Message Storage Module for processing. It retrieves line table entries from the Message Storage Module or elsewhere to do character comparisons as a part of its processing. If a message is found invalid, it notifies (passes an output port designator) the Routing Module for return of the message to the originator, and appends a notation to the message in the Message Storage Module relating to the reason for message rejection; in some cases it may notify the Routing Module for output to the Traffic Service Position (TSP). If a message is found valid, the Validation Module notifies the Routing Module for transmission to each valid addressee.

The Validation Module processes messages in order of their Date/Time of receipt, within each precedence category. It scans a section of global memory ("mailboxes") reserved for intermodule communications, finding pointers to messages requiring processing. Scanning each message header in turn, it extracts for processing that header with the earliest Date/Time Group, within the highest precedence category.

II. Input

A. Format: 16 bit address that will point to a list of parameters.

B. Input Port: 8-bit parallel data; 16-bit parallel address, 4-bit parallel control lines; "fetch" from common memory.

C. Multiplex Requirements: None.

D. Timing (Rate): Timing varies with application. It will vary from a low of 5.7 char sec asynchronous to a high of 1200 char/sec. In a message processing mode, the Validation Module transfers 30 characters per transaction (a message header); in a scanning mode, the Validation Module transfers 12 characters per scan (mailbox contents). All transfers from common memory are one character at a time, asynchronous.

E. Timing (Sequence): Transfer of data is performed asynchronously upon command of the processor via control lines.

F. Addressing: Two addresses per character transferred are required - one for "fetching" from common memory and one for loading to local memory.

III. Output

A. Format: 16 bit return code and parameter list addresses

B. Output Port: 8-bit parallel data, 16-bit parallel address, 4-bit parallel control lines; load to Line Interface Module.

C. Multiplex Requirements: None.

D. Timing Rate: Same rate as Input. In a message processing mode, the Validation Module has two characters per transaction (mailbox status) output to common memory; in a scanning mode, the Validation Module has no output to common memory.

E. Timing Sequence: See Inputs.

F. Addressing: Two addresses per character transfer - one address for "fetching" from local memory, one address for loading to common memory.

G. Other Outputs: None.

IV. Transfer Characteristics

The validation module locates the message header from the parameter list. Each field is validated as described in the general description. If the header contains invalid information, the validation routine will create a message to be routed back to the originator indicating the cause for rejection and set the return code to some positive value. If the header passes all edits, the return code will be set to zero.

SWITCHING FUNCTION MODULE: ROUTING MODULE, TYPE: PRIMARY

I. General Description

This module receives notification from the Validation Module of the location (in the Memory Storage Module) of messages requiring output service, and arranges for the appropriate Output Service/Code Translation Module to acquire each message for processing.

As the Validation Module retrieves the line table entry for each destination addressee, it retains the identification of the output port and Output Service/Code Translation Module appropriate to that addressee. Upon completion of message validation, the Routing Module receives the following data from the Validation Module:

A. Message location in the Message Storage Module;
B. Appropriate output port indicator(s) and Output Service/Code Translation Module indicator(s);
C. Message Date/Time Group; and
D. Message Precedence.

The Routing Module maintains a queue of messages awaiting output service for each Output Service/Code Translation Module. As it receives the above data for each validated message, the Routing Module must adjust queues to insure that each queue is ordered by Date/Time Group, earliest first, within each precedence class (highest precedence first). Each queue entry contains a message location and an output port indicator. The top entry in each queue is retained in a section of global memory ("mailboxes") reserved for intermodule communications. It is scanned periodically by the appropriate Output Service Code Translation Module, and serves as a pointer to the next message needing output service in that module. The Routing Module must maintain cognizance of the output actions performed on each message. In particular, it must note when all copies of a multiple-address message have been transmitted, so that the Memory Management Module can be notified to deallocate the buffer space in the Message Storage Module which was assigned to that message. The Routing Module must also transmit an alarm to the System Control Module in the event of queue overflow or of message retention in a queue for a period exceeding that specified in System Speed of Service (SOS) requirements.

II. Input

A. Format: 16 bit address of parameter list.
B. Input Port: Microprocessor register (direct from Validation Module).
C. Multiplex Requirements: None.
D. Timing: Virtually no time involved (common register with Validation Module).
E. Addressing: Not applicable.

III. Output

A. Format: 16 bit return code word and parameter list address.
B. Output Port: Microprocessor registers.
C. Multiplex Requirements: None.
D. Timing: Timing varies with application. It will vary from a low of 5.7 char sec asynchronous to a high of 1200 char/sec. Routing Module outputs six characters per transaction (buffer addresses to common memory).
E. Addressing: Two addresses required per character: One in local memory, one in common memory.
F. Other Outputs: None.

IV. Transfer Characteristics

The Routing module locates the destination port addresses from the parameter list. A specific output line number is found for each symbolic destination. This line number along with other information, is placed in the Route Control Word (RCW). The RCW's are then put in the line handler queues for output scheduling. (The queue addresses having been provided in the parameter list.)

SWITCHING FUNCTION MODULE; MESSAGE STORAGE MODULE, TYPE: SECONDARY

I. General Description

After a message has been accepted from a loop or trunk, and has experienced code translation and reformatting as necessary, it is passed to the Message Storage Module. The Message Storage Module retains the message from the time it is released by an Input Service/Code Translation Module until it is accepted by the appropriate Output Service/Code Translation Module (or in the case of multiple-addresses, until it is accepted by all the appropriate Output Service/Code Translation Modules). During this retention, portions of the message will be released to the Validation Module and the Routing Module for those processes. Also, all or part of the message will be released to the Message Journaling Module for historical record-keeping.

Depending on the volatility of the message contents, as well as the position of the switch in the switch network*, message retention may be extended to the point where delivery has been confirmed to the destination terminal (even if it is a subscriber to a different switch). In cases where the message cannot be accepted by the appropriate Output Service / Code Translation Module, all or a portion of the message will be delivered to the Output Service/Code Translation Module appropriate to the Traffic Service Position (TSP).

As a type "Secondary", the Message Storage Module may also contain some or all of the program instructions used by other modules. It will contain tables of line-by-line classmarks and statusmarks, as well as routing tables.

It is estimated that 4K words of memory will be sufficient for message storage.

II. Inputs

A. Format: Character strings (messages), 7-bit ASCII with one parity bit per character; average string length 320 characters, maximum string length 1280 characters, in block of 80 characters.

B. Input Port: Interface to architecture.

---

*For example: Whether the message represents loop-to-trunk, trunk-to-trunk, or trunk-to-loop traffic for the switch.

C. Multiplex Requirements: None implicit; actual implementation requirements depend on architecture.

D. Timing (rate): Must be able to accept characters and storage addresses at least as fast as architecture supplies them; actual timing depends on component choices; (average) less than 1 input word per output word.

E. Timing (sequence): Address and input enable signals required no later data signals; simultaneous or sequential arrival may depend on architecture; may depend on component choices.

F. Addressing: One address required per data character input; word length dependent on memory size and mapping technique used.

III. Outputs

A. Format: Same as Input; same maximum character string length; average character string length smaller (200 characters) since both complete messages and partial messages (headers, etc.) are output.

B. Output Port: Same as Input Port.

C. Multiplex Requirements: None implicit; all or portion of each message to be output at different times for different processing.

D. Timing (rate): Must be able to supply characters up to rate at which architecture accepts them; see Inputs.

E. Timing (sequence): Character output accomplished after receipt of address and output enable signals; timing dependent on component choices.

F. Addressing: One address character per output character.

G. Other Outputs: Dependent on architecture and implementation; none implicit.

IV. Transfer Characteristics

A. Throughput Delay: Total of original storage time, recall times for output to various modules for processing, recall time for output to appropriate Output Service/Code Translation Modules), and recall time to output to Output Service/Code Translation Module appropriate to TSP. May also include wait time until confirmation of final delivery. Because the processors attached to the memory are slow (5.5 μsec cycle time) a slow memory is satisfactory.

B. Scaling: 1.1 to 2.5 (approx.) output words per input word.

C. Fault Detection: Error detection and correction (EDAC) may be added, and be transparent to the rest of the system, by adding the properly coded bits to each word of data.

GENERALIZED SECONDARY FUNCTIONS

I. Initialization

All of the actions necessary to begin processing messages are included in the initialization function. The initialization tasks can vary greatly between different switch implementations. These tasks may be handled either manually by the operator or automatically under software and/or hardware control. The number of tasks performed will generally increase as the amount of switch flexibility grows. For a very rigid system initialization may be simply applying power to the switch. In the more general case, however, initialization would involve performing specific actions needed to set-up each of the other functions necessary to the proper functioning of the switch. For example, before message routing can operate properly, the route tables must be built and verified. System Control must be aware of all the input and output ports he is controlling and of their current status. Buffer space must be provided for messages. The current memory configuration must be supplied to Memory Management. These along with other implementation dependent tasks prepare the switch to begin processing messages. Once all of the necessary initialization tasks have been performed, processing is begun by System Control.

II. System Control

The means by which the other functions are accomplished is the function of System Control. There may be a number of functions performed in accomplishment of switching a message. To determine the orderly processing of these functions System Control must maintain certain housekeeping activities. An accurate status of the switch and an exact configuration of the input/output ports must be maintained. The input/output queues must be adjusted to handle the message traffic load. The results of these housekeeping activities allow System Control to determine the next function to be performed. If such a determination cannot be made, control will be passed on the Error Recovery function.

III Memory Management

All memory allocation and deallocation is the function of Memory Management. Requests for service are generally received from only System Control. These requests are processed successfully or are rejected for some specified reason. A memory reorganization routine is usually included to aid in more efficient memory usage. This function is a simple but very important part of the message switch.

IV. Error/Recovery

Error detection and the corresponding corrective actions make up the Error/Recovery function. Error detection is initiated whenever System Control encounters any unanticipated conditions. These conditions are viewed by the detection routines to ascertain what error has occurred. If possible, the recovery

routines will correct the situation before returning control to the system. As a result of corrective actions taken, the system may continue operating at full capacity or at some degraded state. If continuation is impossible, the alarm system is used to signal a request for manual intervention. In most implementations it is hard to make a clear division between System Control and Error Recovery.

V. Diagnostic/Test

The auxiliary function used to examine and analyze the message switch is Diagnostic/Test. This function is unique from other switch functions, since it is not necessary for proper operation. It is included to ensure the timely correction of switch malfunctions by helping to localize the problem and to verify proper switch operation. The Test function is used on a regular basis to exercise all the switch functions and ensure they are functioning properly. The diagnostics function is available to aid the servicing technician to locate and correct faults in the switch after an alarm has been activated by Error/Recovery. Typically the message switching functions must stop while Diagnostic/Test is being utilized.

APPENDIX K

SIMULATION PROGRAM FLOWCHARTS

CONTENTS

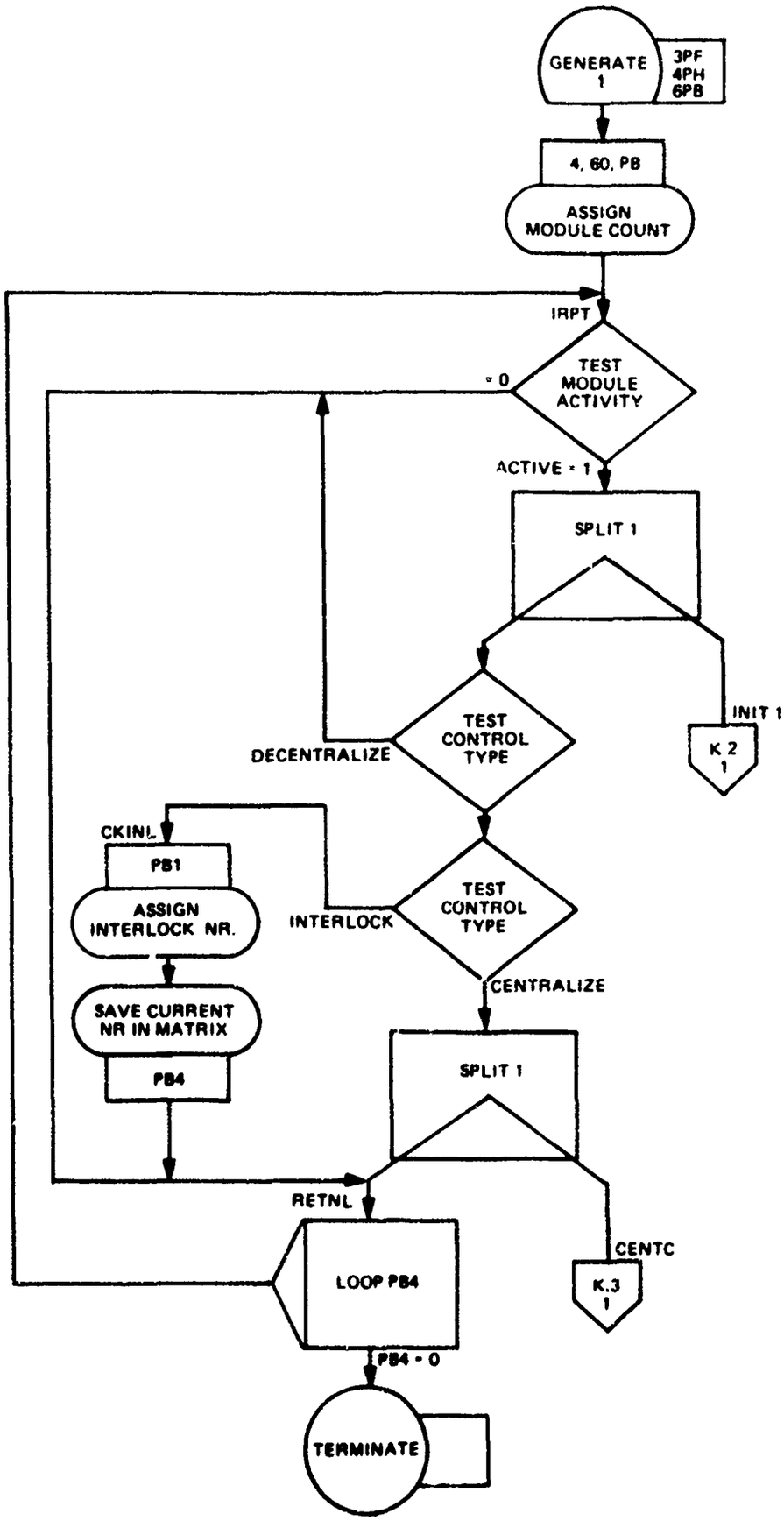
BRIEF DESCRIPTION

The flowcharts that follow are a representation of the general signal processing and switching model studied under this contract. The symbols are GPSS blocks that relate to the major functions of the model. The flowcharts are interconnected by off-page connectors, which identify the figure number and connector number.

Figure K.1 generates the specified data control type for each of the active modules in the system. Figures K.2 and K.3 show the logic for control generation for decentralized and centralized control respectively. The control transactions remain dormant until the specified time has expired whereupon the input and/or output routines (figures K.4 and K.5) are activated.

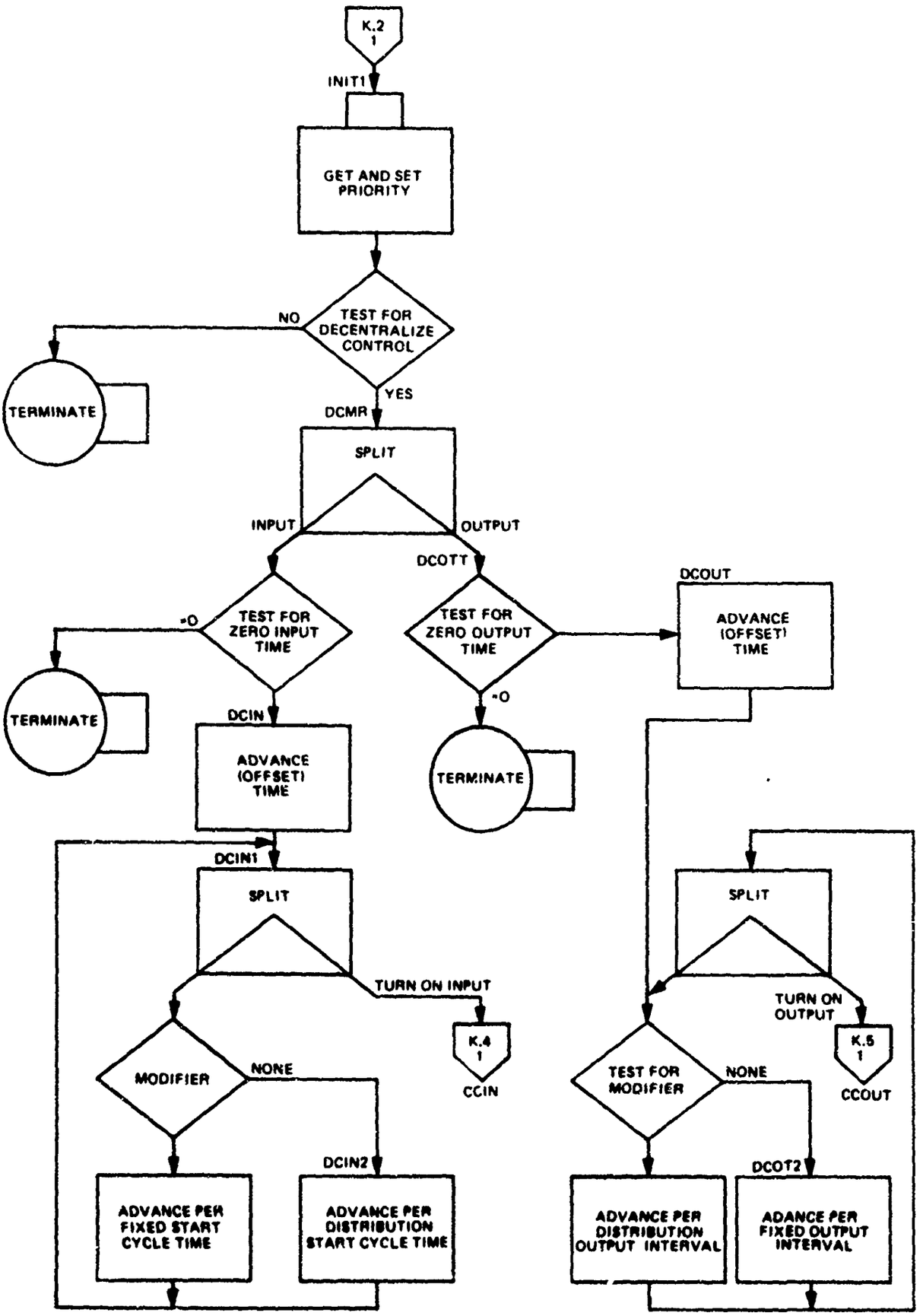
Figure K.6 is a block diagram of the input/output routine used when bus transfer is accessed.

Figure K.7 is a separate routine that monitors the run time of the simulator and terminates the simulation run when time has expired. It additionally tallies all the statistics accumulated during the run.



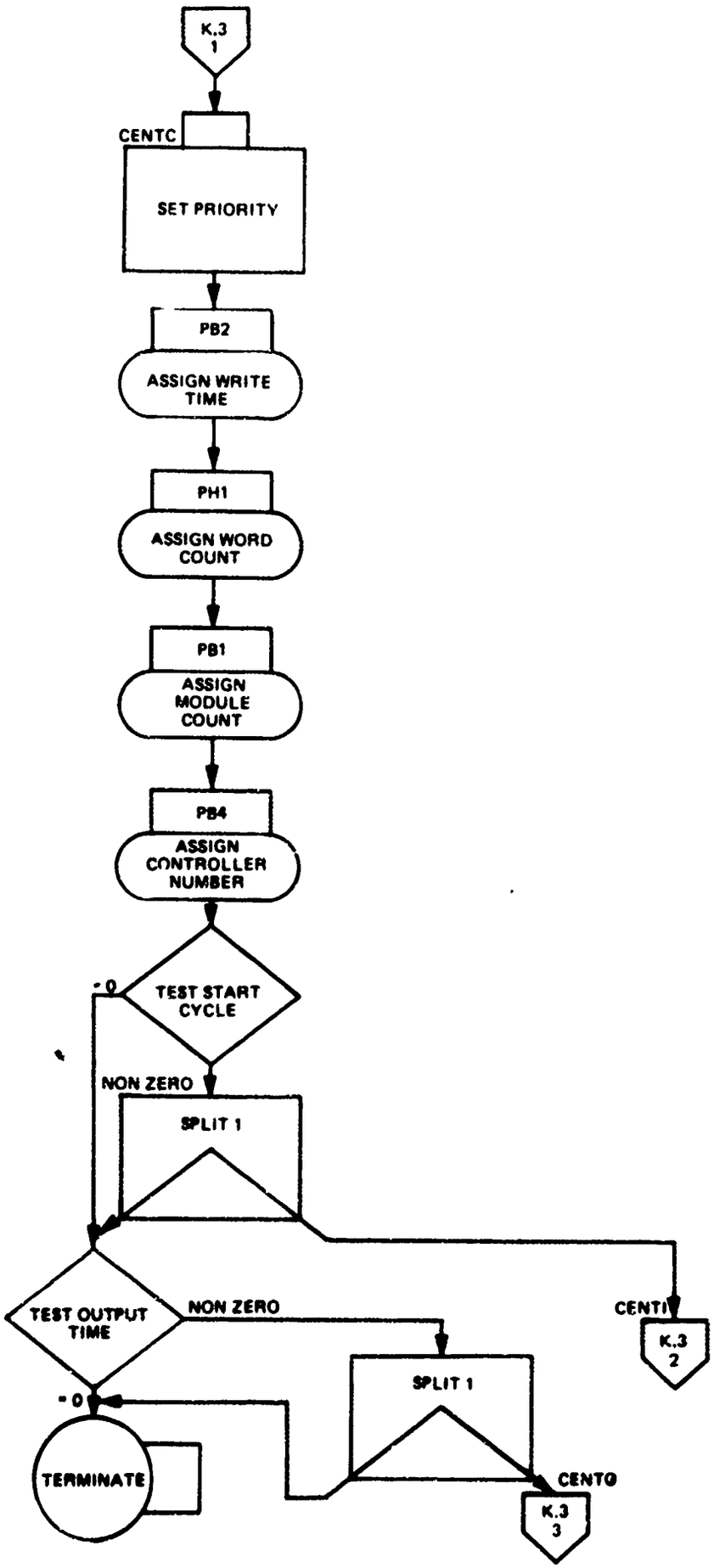
21877-19

K.1 Module Initialize Loop.



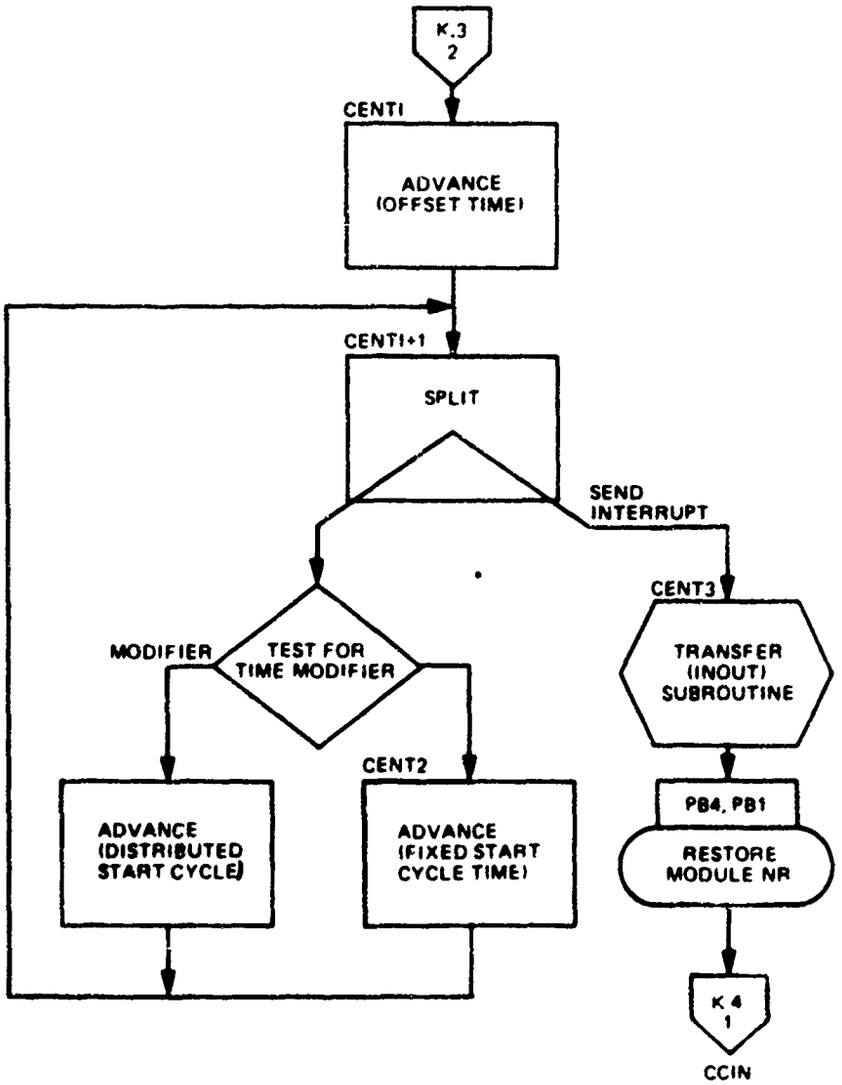
K.2 Decentralize Control

21877-20



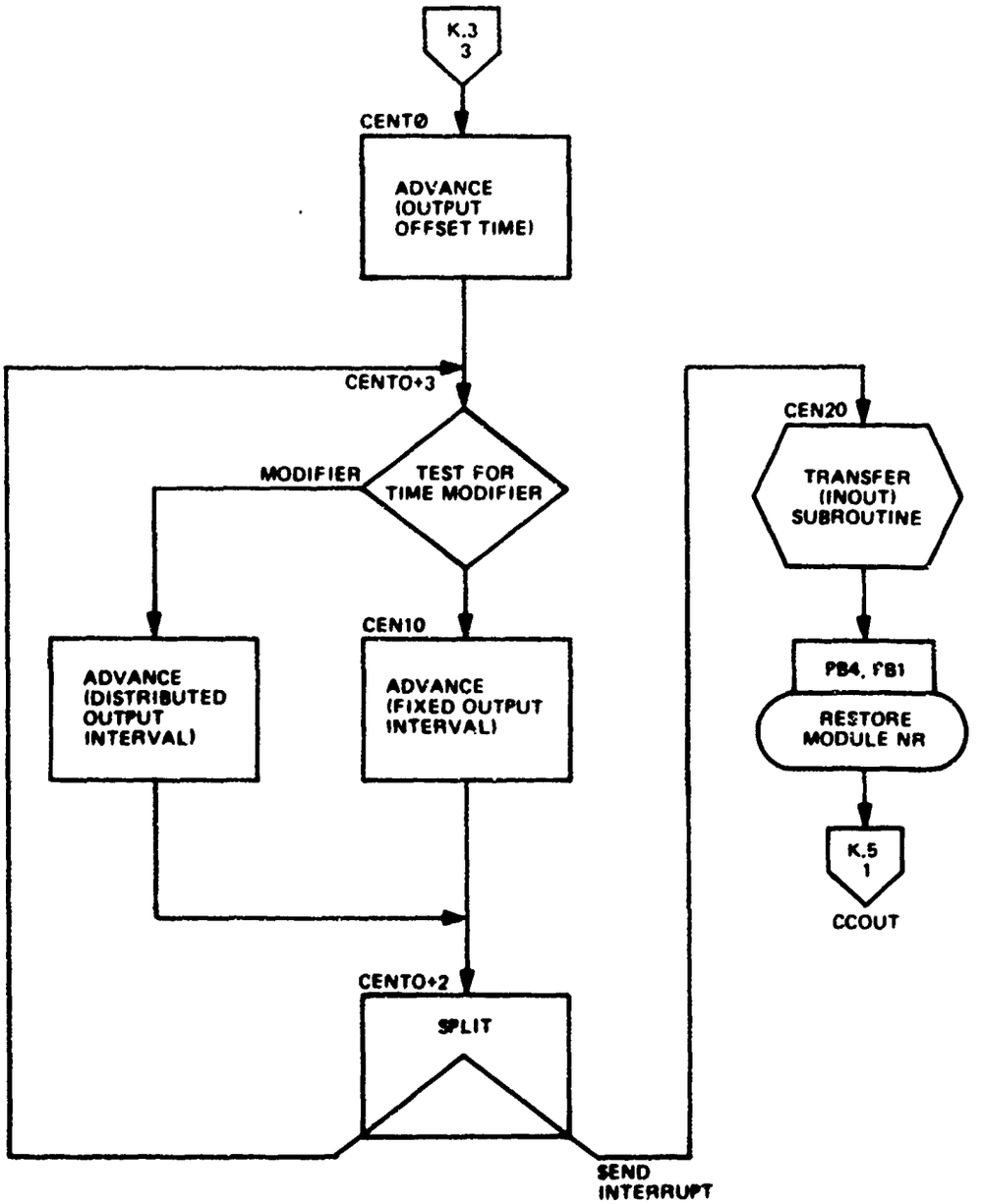
21877-21

K.3a Centralize Control Setup



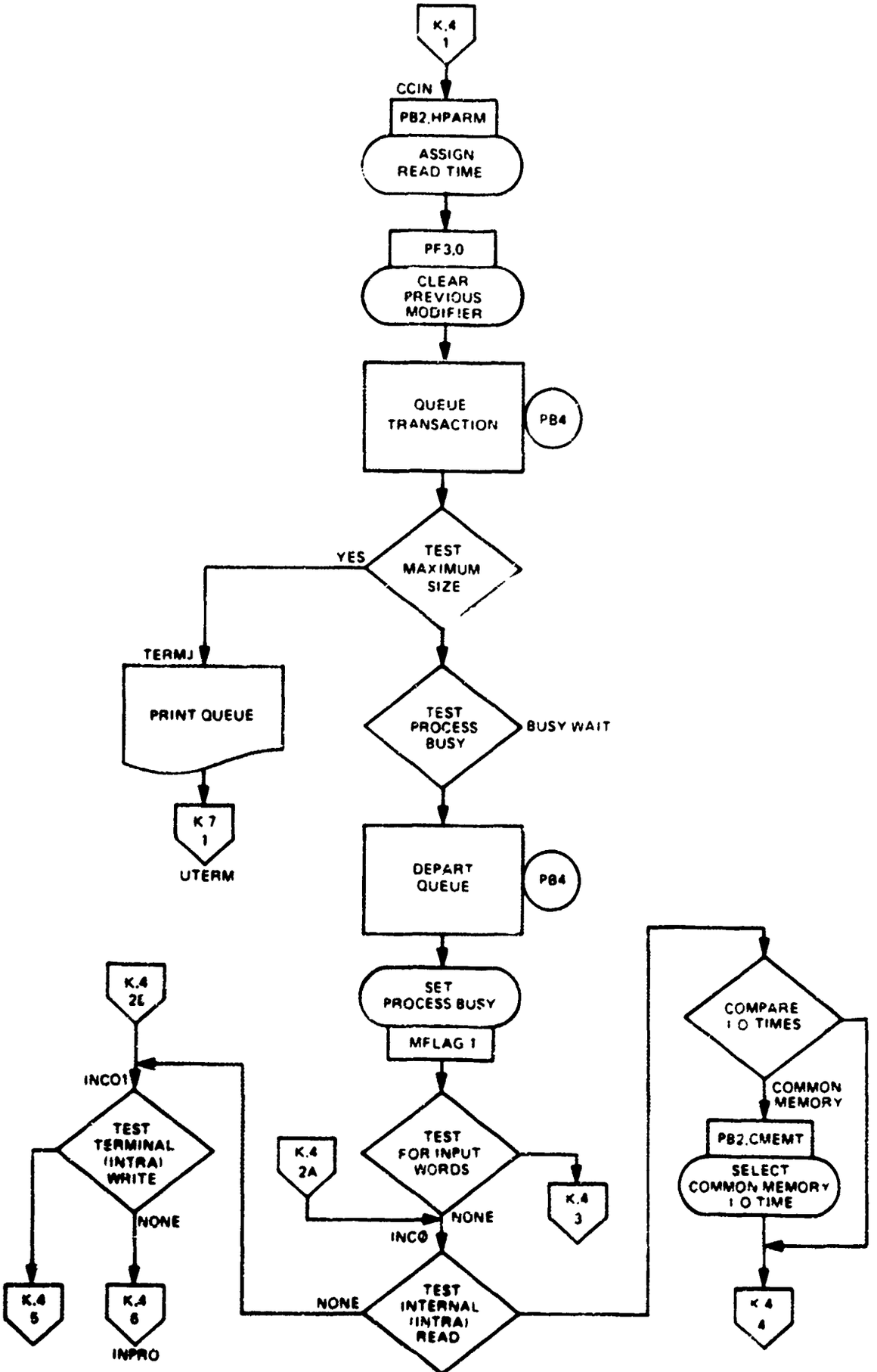
21977-22

K.3b Centralize Input Start Control

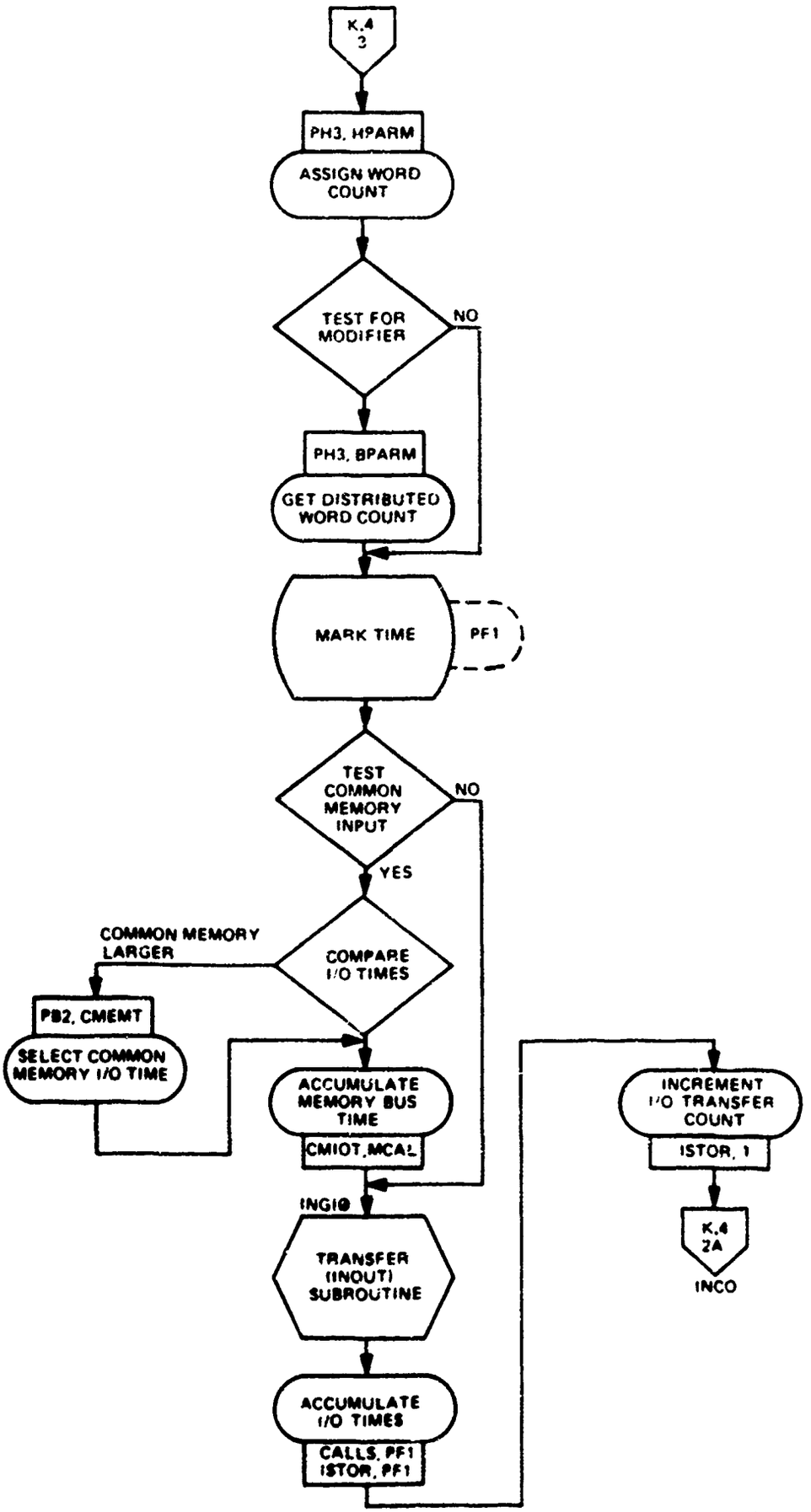


21877-23

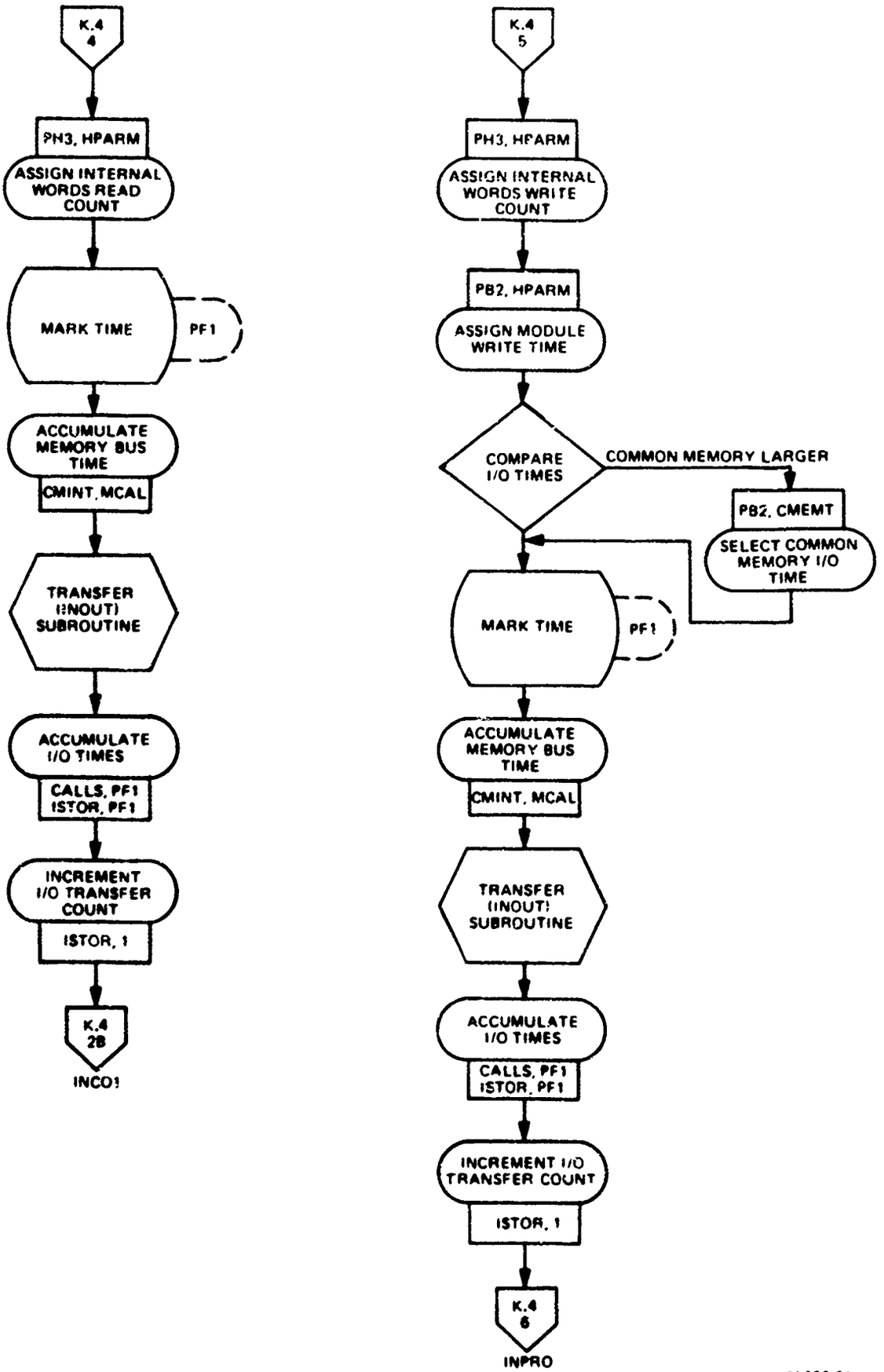
K.3c Centralize Output Control



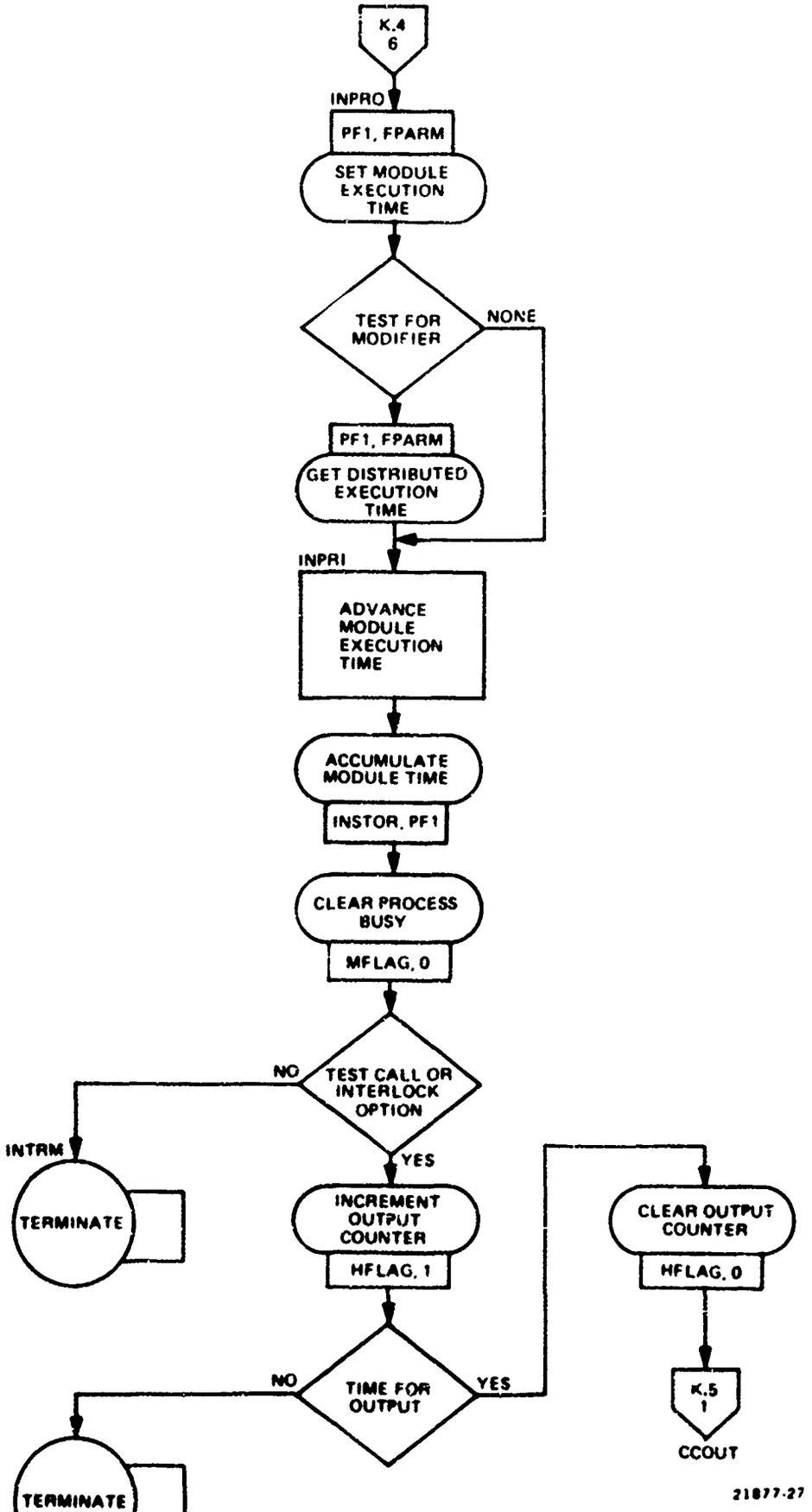
K.4a Input Service



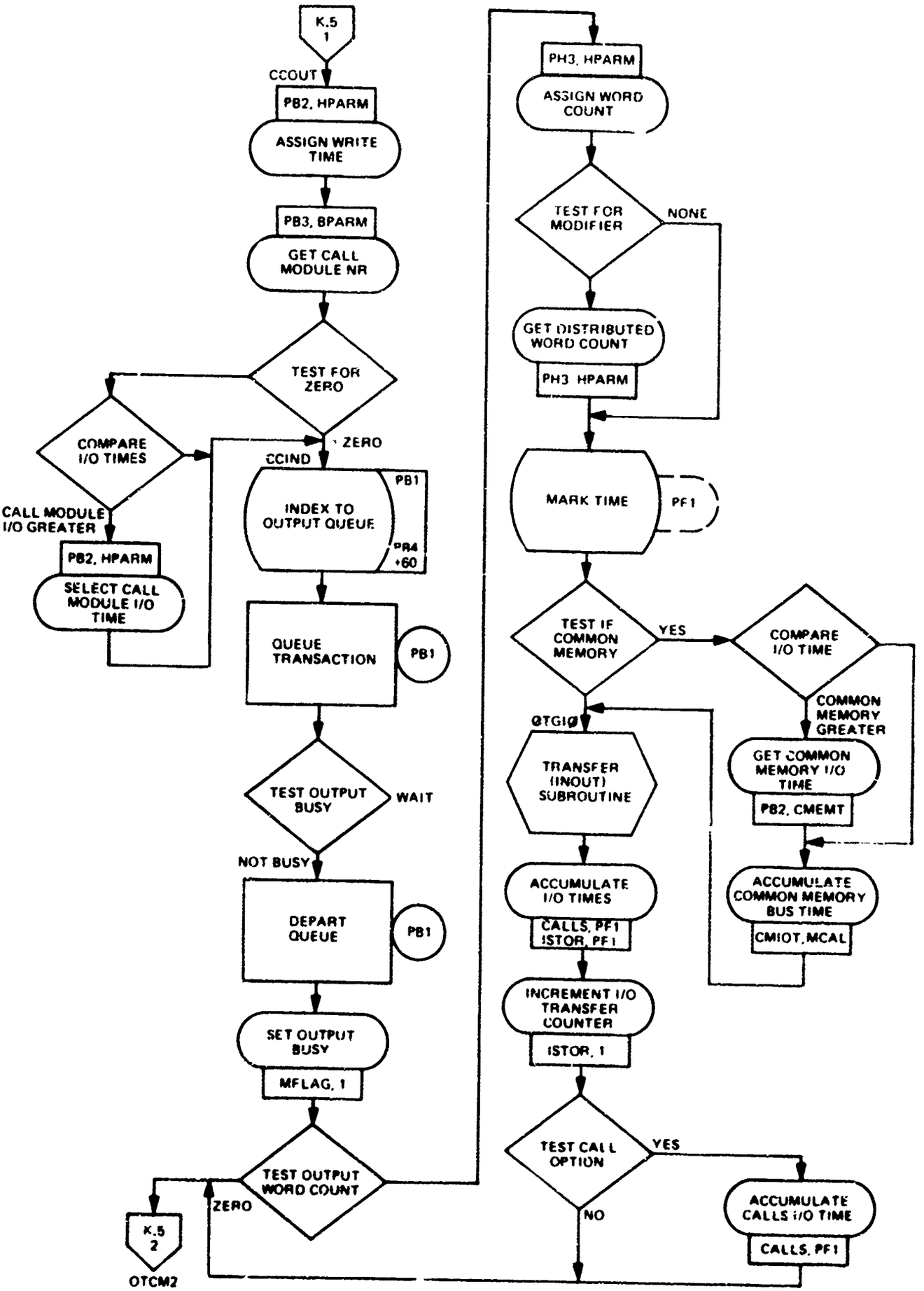
K.4b Input Service (continued)



K.4c Input Service (continued)

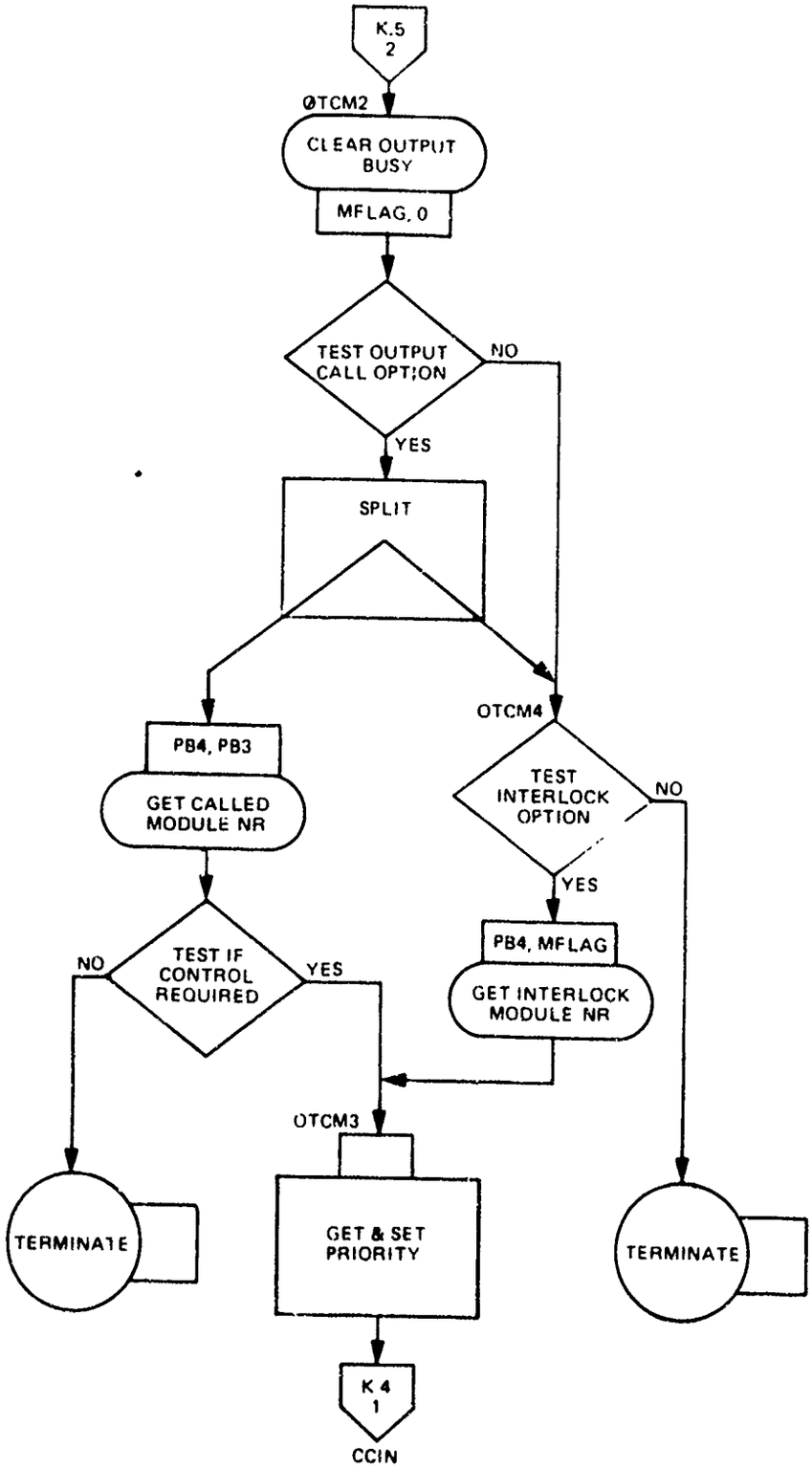


K.4d Input Service (continued)



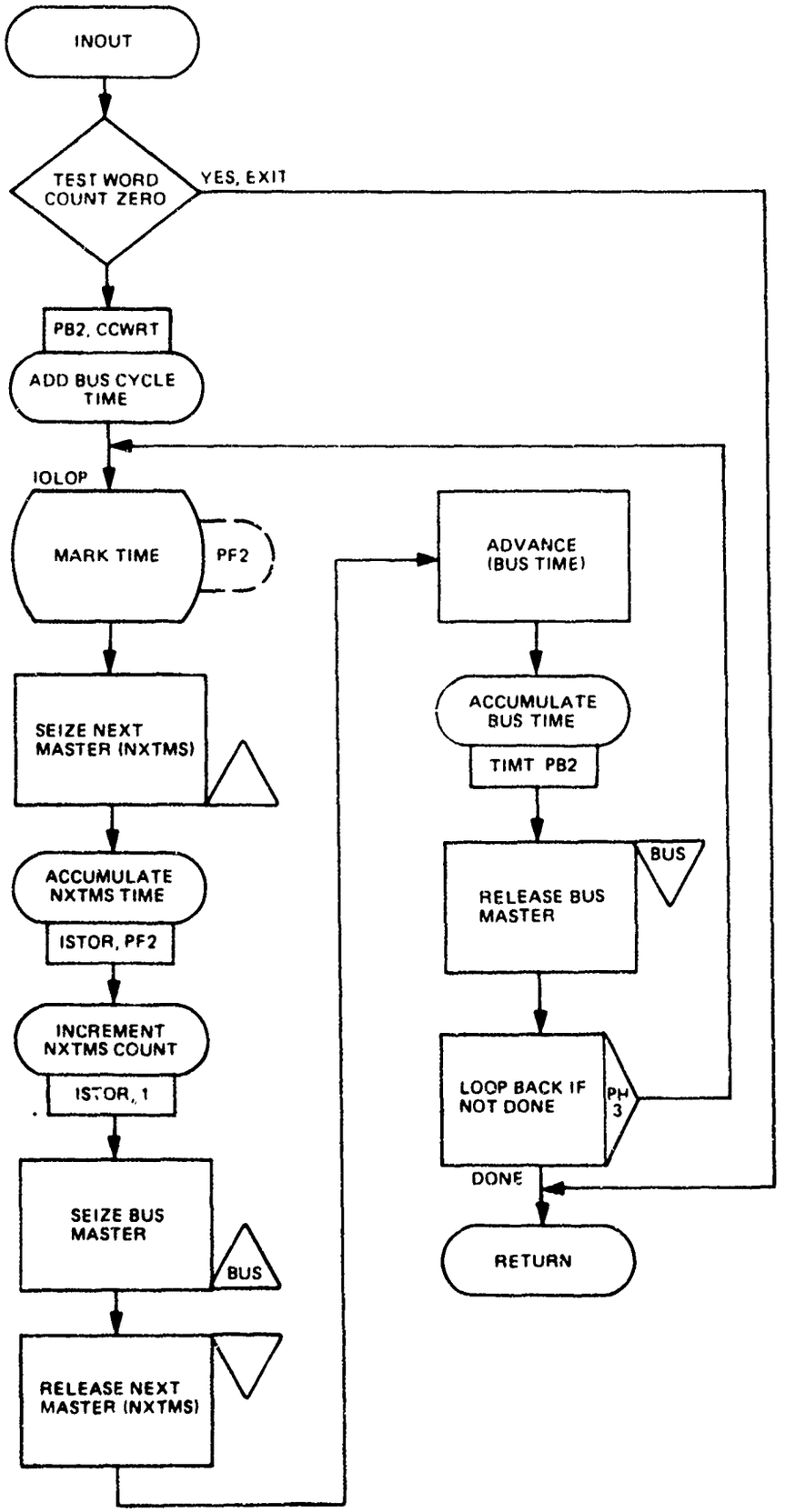
K.5a Output Service

21877 28



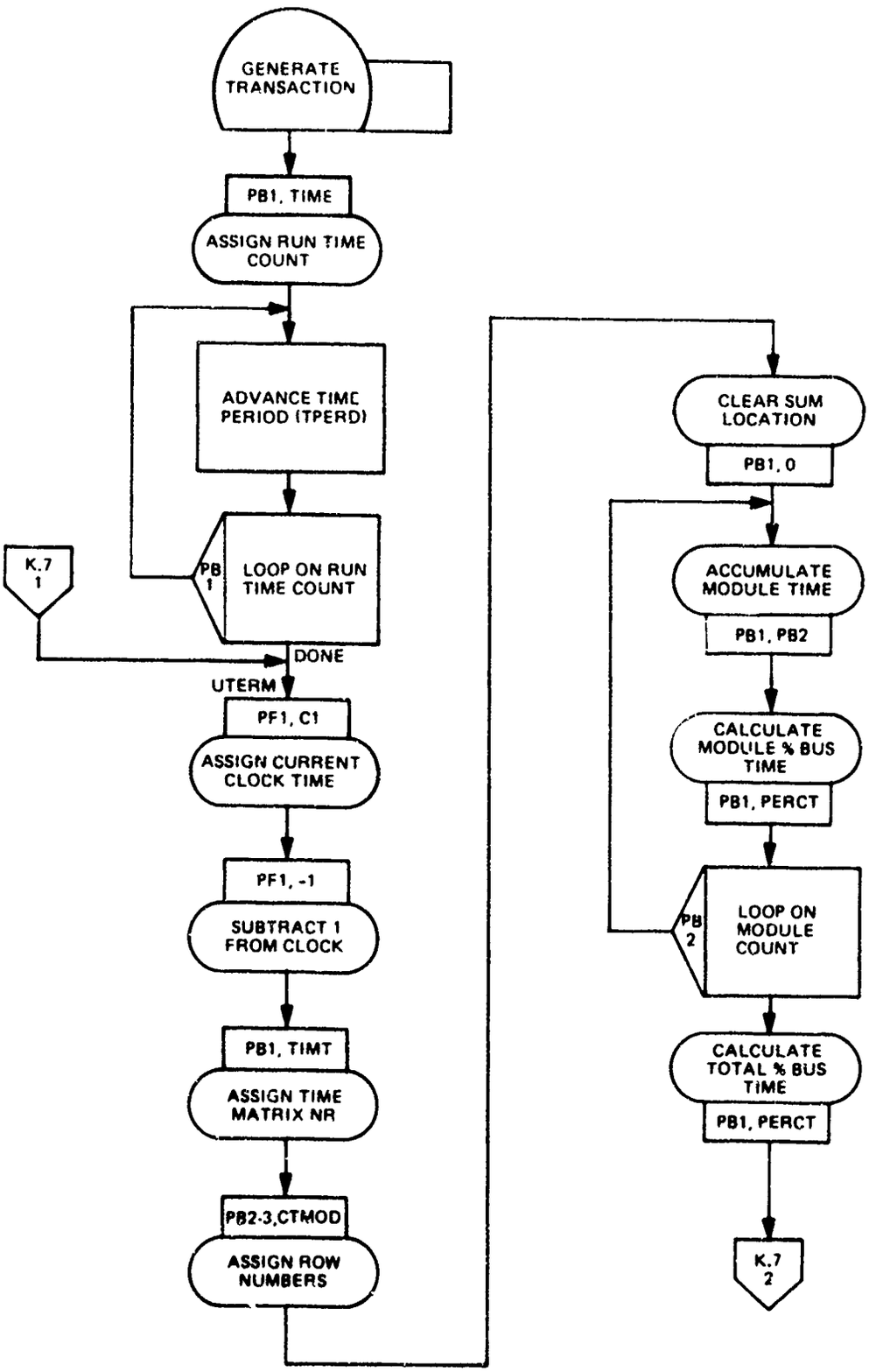
21877 29

K.5b Output Service (continued)



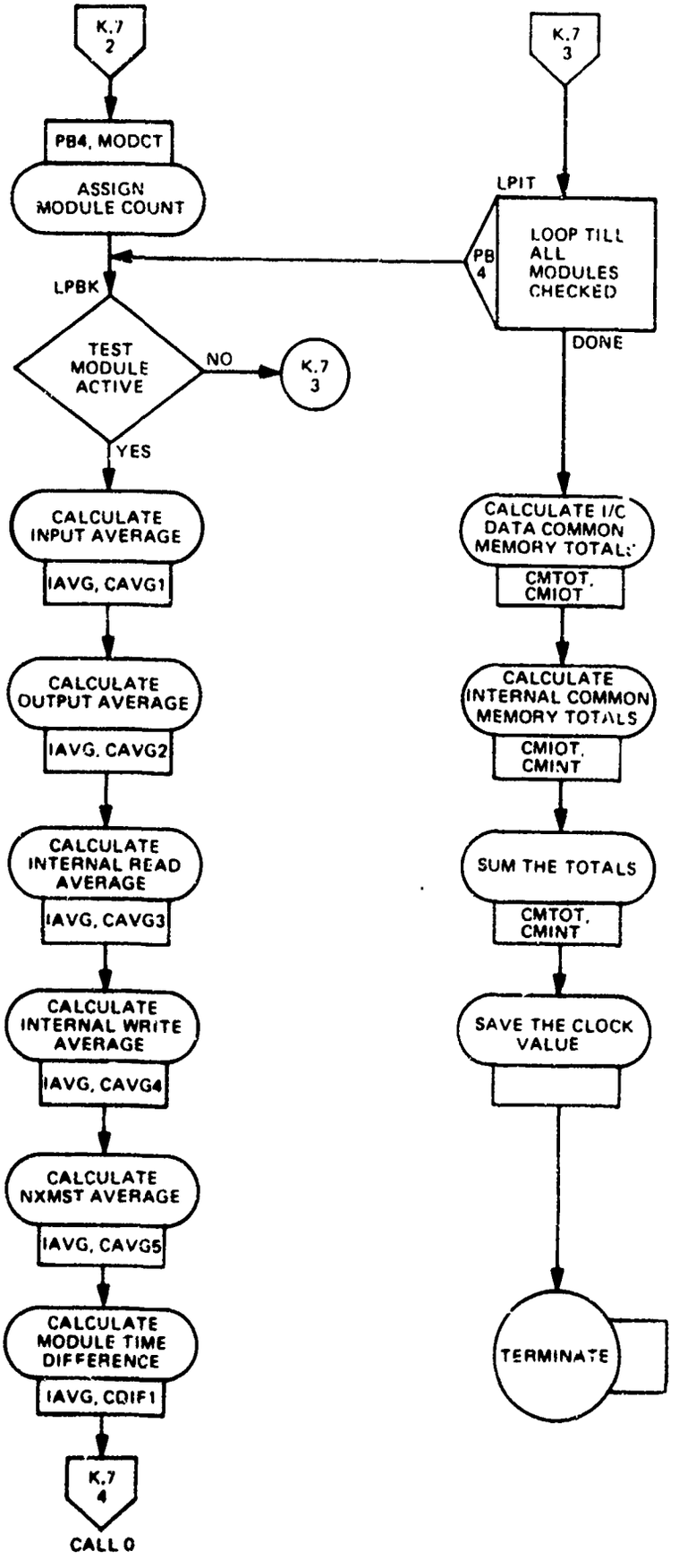
21877 30

K.6 Bus Input/Output Transfer



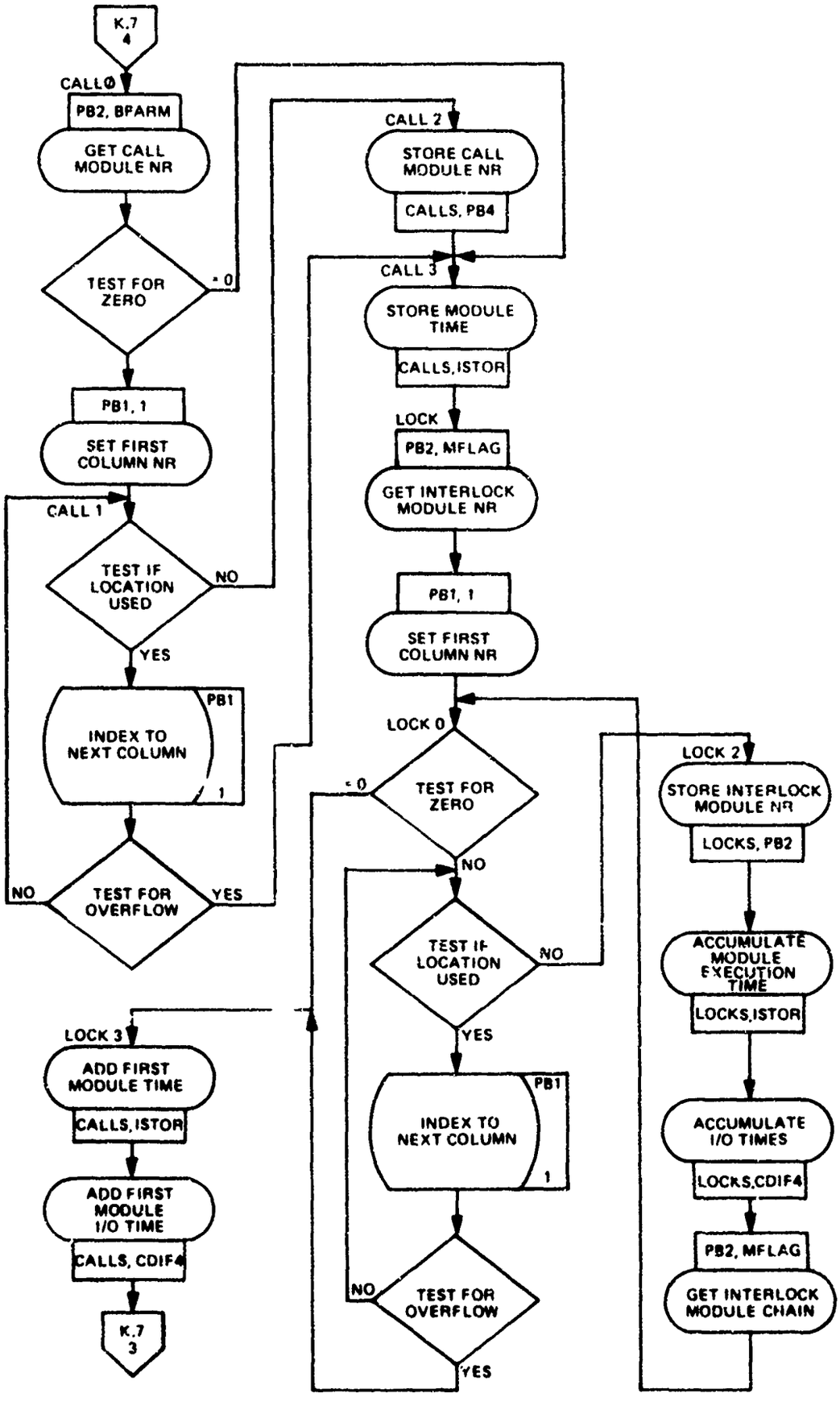
21877-31

K.7a Run Time Monitor and Statistic Accumulation



21877-32

K.7b Statistics Accumulation



K.7c Statistics Accumulation (continued)

APPENDIX L

COMPUTER LISTINGS AND RUNS
(Refer to Volume 3)

APPENDIX M

LANGUAGE SELECTION

LANGUAGE SELECTION

1. Background

A. Collins Avionics Division made an in-house study which resulted in the paper titled "A Proposed Long Range HOL Plan for Collins Radio", WP 3954 dated March 17, 1976.

The applications area was defined as avionics, communi ·ation switching and communication products utilizing imbedded processors.

Languages studied were PL/I, PL/M, PL/M6800, MPL, PL-MIPROC, FORTRAN, BASIC, JOVIAL J3B, AED.

The results of the study indicated that a subset of PL/I would be most effective.

B. Collins Government Telecommunications Division made an in-house study based on the DoD initiatives presented at an AIAA Conference on April 5–6, 1976.

The applications area was defined as signal processing, message handling and device control in the secure communications field for equipments utilizing imbedded processors.

Languages studied were CMS-2M, Pascal, Concurrent Pascal, Euclid, PL-I, AED, CORAL 66, FORTRAN, IFTRAN.

The context for this study was the DOD Tinman document which set forth a number of requirements and desirable features. The objective was to anticipate the characteristics of the language which would finally result from DOD's initiatives, and to select a similar existing language for near term secure communications applications.

The results of the study indicated that the language should strongly resemble Pascal with extensions directed toward:

explicit real and complex arithmetic
multi-tasking and resource management.

The language design problems involved in these extensions is clearly non-trivial if machine dependencies are to be avoided, and compiler implementation would be costly.

The conclusion was that no available language was close enough to the mark to be selected for use, and that the selection should be delayed until the results of ongoing work in various places could be evaluated.

A side result of the study was the recognition that the requirements of formal program verification techniques should have a strong influence on the language design, particularly in the applications area of secure communications. It is not expected that these techniques will be widely applied in industry for a few years, but they will then probably become contractual requirements.

C. At the beginning of the present study, the report on COL, DCEC Report R-3261 dated March 1976 was brought to our attention. This report contains a section on language comparisons.

The applications area was defined as communications programming and systems programming.

Languages compared were FORTRAN IV, JOVIAL J3, PL/I, C, BCPL, IMP, ESPL-1, TPL2, BLISS, Pascal.

A conclusion of the report was that modification of an existing language to satisfy DCA's requirements would not be a cost effective alternative to development of a new Communication Oriented Language (COL).

The design of COL incorporates the worthwhile features of Pascal and has extensions directed toward multi-tasking and resource management. This agrees substantially with the results of study (B) above.

2. **Briefly Considered**

Some languages were considered briefly and dropped because they:
satisfied too few of the criteria given in section 3.3.5 of the main report
violated too many
were poor prospects for rework or extension.

In this category were PL/M, PL/M6800, MPL, JOVIAL, PL-MIPROC, BCPL, IMP, ESPL-1, C, TPL2, BLISS, BASIC, CMS-2, COBOL, EUCLID.

3. **Seriously Considered**

A. Coral 66: This language is mandated by the British Ministry of Defense and as such was of special interest in this applications area. The language specification dated May 1970 shows it to contain many of the needed features. However, the treatment of a Fortran-like Common, reliance on implementation-dependent operating systems, and other problems involving escapes to machine code, led us to the conclusion that although it overlapped other candidates such as PL I and SPL I to a large degree, there were disadvantages without significant offsetting advantages. The matter of language control being shared between two Defense establishments also brings in a new set of pro and con arguments which are believed to be outside the scope of this task.

B. FORTRAN: This language, particularly in its structured variants, must be seriously considered because of its unique portability. The implementation needed for portability requires a pre-processor which turns block structured symbolic input into normal compilable Fortran source code. Many users claim ease of use and production of efficient object code. However nothing more can actually be done that cannot be done in Fortran directly. The well known limitations on data structures; potential for problems with Common constructs; heavy dependence on a run-time machine-coded library to connect with machine features and the attendant overhead' all these considerations lead to the conclusion that PL/I should be considered rather than Fortran.

As pointed out in the main report, however, Fortran will probably remain an important factor in the generation of portable compilers, assemblers, linkers and emulators.

C. Pascal: This language has received a great deal of attention because of its clean design and ingenious data structures, strong typing and the like. However the clean design is partly a consequence of dismissing certain features which are needed in this application area, some of which are addressed by Concurrent Pascal.

D. Concurrent Pascal: The sequential language Pascal is extended to include programming tools called processes and monitors. These are used to enable efficient operating systems to be written with multi-tasking and resource control. It is under a continuous state of development in the academic and commercial fields where other extensions involving arithmetic have also been installed. As finally extended this language will be a strong contender for serious study. Meanwhile, for the purposes of the current task, this language will be considered to overlap with COL to the extent that we can proceed as if COL subsumes it in principle. That is, the same processes and monitors can be built even if it is necessary to use the Machine-Like-Code feature of COL to do it.

4. Selected for Comparison

It became apparent during the study that none of the HOL's being reviewed responded fully to the criteria developed. The notion of a "composite-best" language was considered, against which the criteria could be tested for reasonableness and to establish the size of the gap between required and available features. As this would have required generating yet another language definition, something the world already has plenty of, it was decided instead to select about three of the leading candidates and rate them against the criteria as a collection.

PL/I was selected because it satisfied most of the criteria with one outstanding violation, strong type checking. A consideration which is not on the list of criteria but is nevertheless very significant is PL I's wide implementation and acceptance. On balance, the selection of PL I is believed to be justified, particularly if we allow the following approach

- for this applications area, we can easily get along with a proper subset of PL I.
- with careful subsetting, a PL/I compiler can be implemented which will produce demonstrably efficient object code, unlike some of the large scale general-purpose-processor implementations.

SPL/I was selected because it meets most of the criteria and in addition is approved for use under DOD Instruction 5000.31.

COL was selected because it meets most of the criteria and has the best treatment of Architecture Oriented Code (or as it is referred to in the COL specification, Machine-Like-Code).

Each of the three languages has most of the features which are to be expected in a current-day language, in which detailed comparisons of languages tend to produce "distinctions without a difference". These include:

- Boolean and logical data types
- Variations of DO constructs
- Variations of IF constructs
- Multibranch conditionals, with satisfactory CASE constructions for SPL/I and COL and some less tidy equivalents in PL/I
- Indexed arrays
- Other data structures with associated pointers
- Dynamic storage assignment/free mechanisms

In other words, although the above features differ in detail and power it is difficult to find examples in the application area being studied that would require a change to the language.

In Figure 3.3-6 of the main report, some of the YES/MLC/NO entries in the table require further comment. See Table M-1.

Table M-1.

	PL/I	COL	SPL/I
1. HOL Mechanisms for parallel task control	TASKS EVENTS LOCK UNLOCK PRIORITY OPTION	REGION IFLOCKED LOCK UNLOCK ELSE RETRY	PROCESSES SIGNALS REQUEST RELEASE PRIORITY ATTRIBUTE
4. Complex arithmetic	Real: Real	Note 1	Integer: Integer Fraction: Fraction Real: Real
5. Character handling	CHARACTER	Note 2	STRING (N)
6. Bit manipulation	BIT-string	INTEGER (BIT 3)	BIT (N)
7. Precision control	Note 3	Note 3	SINGLE DOUBLE
10. Type checking	Note 4	Note 4	YES

NOTES

1. The type declarations of COL allow two reals to be paired, but this begs the question of needing complex arithmetic syntax of the kind A: = B * C + D
2. The type CHAR is used as a pseudonym for LOGICAL data of a particular (implementation-dependent) size and is for programmer convenience only. Type-checked as LOGICAL.
3. PL/I: fixed precision. Cumbersome for fractions because of left truncation on multiply
   COL: INTEGER (A N) where A signifies BIT, BYTE or WORD, and N = 1, 2, 3, . . .
4. PL/I is the reverse of strongly typed, to put it mildly, because of built-in conversions that may also be implementation dependent Puts the burden on the programmer.

   COL is strongly typed not only in the compiler but the proposed linker will perform type checking when linking separately compiled modules.

APPENDIX N

BIBLIOGRAPHY

APPENDIX N

BIBLIOGRAPHY

The following bibliography has been organized in four sections: (A) Architecture and Implementation; (B) Design Methodology; (C) High Order Languages; and (D) Standards. Despite this attempt at categorization, overlap does exist between sections (A) and (B), and topics of interest in the general area of systems design propositions will be found in both sections.

A. Architecture and Implementation

1. Allen
   "Computer Architecture For Signal Processing"
   Proc IEEE, Volume 63, N 4, April 1975, pp. 624-633

2. Baer, J. L.
   "Multiprocessing Systems"
   IEEE Transactions on Computers, C-25 No. 12, pp. 1271-1277
   This paper surveys the state-of-the art in the design and evaluation of multiprocessing systems.

3. Bass
   "A Peripheral-Oriented Micro-Computer System"
   Proc IEEE, Volume 64, No. 6, June 1976, pp. 860-872.

4. Bell
   "More Power By Networking"
   IEEE Spectrum, February 1974, pp. 40-45.

5. Bell, C. G. and A. Newell
   Computer Structures: Readings and Examples
   McGraw-Hill, New York, 1971.

6. Bell, C. G., J. Grayson and A. Newell
   Designing Computers and Digital Systems
   Digital Press, Digital Equipment Corporation; Maynard, Mass. 1972.

7. Bell, C. G. and P. Freeman
   "C.ai — A Computer Architecture for AI Research"
   Fall Joint Computer Conference, 1972, pp. 779-790.

8. Davis, R. L., S. Zuker and C. M. Campbell
   "A Building Block Approach to Multiprocessing"
   Spring Joint Computer Conference, 1972, pp. 685-703.

9. Deans, J. L., D. G. Ostrander and S. L. Quilici
   "CAPS Message Processor System Design"
   Collins Internal Working Paper No. 9509, December 1976
   Investigates the feasibility of using the Collins Adaptive Processor (CAPS) in the design of a small tactical message switch.

10. Deerfield
    "Architecture Study of a Distributed Fetch Computer"
    NAECON, 1971, pp. 214-217.

11. De Mori, Rivoira, Serra
    "A Special Purpose Computer for Digital Signal Processing"
    IEEE Trans On Computers, Volume C-24, No. 12, December 1975, pp. 1202-1210.

12. Foster, James R., Jr., 1st Lt. USAF
    "Development of a High Order Language Architecture"
    NEACON "71" Record, pp. 201-205.

A. Architecture and Implementation (continued)

13. Fuller, S. H. and D. P. Siewiorek
"Some Observations on Semiconductor Technology and the Architecture of Large Digital Modules"
Computer, October 1972, Vol. 6, No. 10, pp. 15-21.

14. Gilhousen
"A Multi-Stack Microprocessor for Satellite Modems"
NTC, 74, pp. 543-547.

15. Gold
"Parallel and Sequential Trade-off in Signal Processing Computers"
NTC, 1974, pp. 491-495.

16. Gold, Lebow, McHugh, Radar
"The FDP, A Fast Programmable Signal Processor"
IEEE Trans on Computers, Volume C-20, No. 1, January 1971, pp. 33-38.

17. Harshman
"Architecture of a Programmable Digital Signal Processor"
NTC, 74, pp. 496-500.

18. Heart, F. E., S. M. Ornstein, W. R. Crowther, and W. E. Baker
"A New Minicomputer/Multiprocessor for the ARPA Network"
Proceedings AFIPS National Computer Conference, Vol. 42, 1973. The IMP for the ARPANET is described as being a multiprocessor system connecting seven processor busses, two memory busses, and an I/O bus using a distributed matrix switch.

19. Hornbuckle, Anlona
"The LX-1 Microprocessor and Its Application to Real-Time Signal Processing"
IEEE Trans on Computers, Vol. C-19, No. 8, August 1970, pp. 710-720.

20. Koczela, Wang
"The Design of a Highly Parallel Computer Organization
Autonetics Report No. X9-310/301, 1969.

21. Maslanka, R. E., R. C. Hollis and D. R. Wilson
"Microprocessor Based Control System"
GTE Automatic Electric Technical Journal, October 1976. This article describes a microprocessor application for the emulation of a real-time minicomputer controlled data acquisition system.

22. Matney, R. M.
"A Dual Processor System for Store and Forward Message Switching"
IEEE Intercon 1974, pp. 19/2-1 to 19/2-9. This paper describes a dual processor concept of message switching.

23. Merkel, Chen
"Microcomputer Application to a Spread Spectrum Frequency Hopping Modem"
NTC, 74, pp. 536-542.

24. North Electric Company, Government Products Division
"Interim Technical Report for Communications Processor System"
In accordance with contract F30602-73-C-030, this paper defines a Communications Processor System (CPS) architecture capable of performing both Circuit and Message Switching.

A. Architecture and Implementation (continued)

25. Peterson, E.
"Load-Sharing — An Approach to Using Mini-Computers for Message Switching"
IEEE Intercon 1974. A discussion of load sharing as it pertains to message switching is presented in this paper.

26. Rothmuller, Ken
"Task Partitioning in Programmable Logic Systems"
Computer, January 1976, pp. 19-25.

27. Russell
"CAPS Transfer Bus"
Collins Radio WP3912.

28. Searle, B. C. and D. E. Freberg
"Tutorial: Microprocessors in Multiple Processor Systems"
Computer, October 1975, pp. 22-30.

29. Tinklepaush, N. L. and D. C. Eddington
"Quick and East Design (QED) of Systems Through High-Level Functional Modularity"
NELC Study Program 2175, 28 January 1974.

30. Thompson
"Digital Signal Processor Architecture"
NTC, 74, pp. 501-506.

31. Wecker, S.
"A Building Block Approach to Multi-Function Multiple Processor Operating Systems"
AIAA Computer Network Systems Conference, Huntsville, Alabama (April 16-18, 1973).

32. Wu, Y. S.
"Architectural Considerations of a Signal Processor Under Micro-Program Control"
Spring Joint Computer Conference, 1972, pp. 675-683.

33. Wulf, W. A. and C. G. Bell
"C.MMP — A Multi-Mini-Processor"
Fall Joint Computer Conference, 1972, pp. 765-777. A summary of Carnegie-Mellon University multiprocessor computer system is given in this paper.

B. Design Methodology

1. Coviello, Dr. G. J. and Vena, Dr. P. A.
"Integration of Circuit/Packet Switching by a SENET Concept"
National Telecommunications Conference 75, Vol. 2, pp. 42-12 to 42-17. This paper describes a Slotted Envelope Network (SENET).

2. Esterling, R. and P. Hahn
"A Comparison of Digital Data Network Switching Alternatives"
NTC 75, Vol. 2, pp. 42-8 to 42-11. This paper presents a quantitative comparison of a packet-switched network and a circuit-switched network.

B. Design Methodology (continued)

3. Fischer, M. J. and T. C. Harris
"An Analysis of an Integrated Circuit and Packet-Switched Telecommunications System"
NTC 75, Vol. 2, pp. 42-18 to 42-23. This paper describes a means of multiplexing voice and data traffic in an integrated system.

4. Friend, G. E.
"Selection Criteria for Minicomputer Controlling Communications Switching Systems"
IEEE Intercon 1974. The criteria for selecting a minicomputer for controlling the switching function in a circuit switching and a message switching system are examined.

5. Frish, I., R. Kaczmarck, and B. Occhiogrosso
"7 Steps to Picking the Best Communications Processor"
Data Communications, Nov/Dec 1976, Vol. 6, No. 6, pp. 25-37. An orderly approach to processor selection avoids pitfalls of inappropriate equipment.

6. Jenny, C. J. and K. Kümmerle
"Distributed Processing Within an Integrated Circuit/Packet-Switched Node"
IEEE Transactions on Communications, Vol. COM-24, No. 10, Oct 1976, pp. 1089-1100.

7. McQuillan, J. M., W. R. Crowther, B. P. Cosell, B. C. Walden, and F. E. Heart.
"Improvements in the Design and Performance of the ARPA Network"
Fall Joint Computer Conference 1972. After three years of operation, the ARPA network has added new insight into the problems of computer networks.

8. Miyahara, H., T. Hasegawa, and Y. Teshigawara
"A Comparative Evaluation of Switching Methods in Computer Communication Networks"
IEEE ICC 75, Vol. 1, 6-6 to 6-10. Message, packet and line switching are analyzed by a queueing model.

9. Ornstein, Severo M., and David C. Walden
"The Evaluation of a High Performance Modular Packet-Switch"
IEEE ICC 75, Vol. 1, 6-17 to 6-21. The authors discuss the design of a series of packet-switching systems.

10. Plotkin, I. P.
"Overview of Systems Control"
NTC paper 1976, 22.1-1 to 22.1-3. This paper discusses on-going developments and compares strategic and tactical aspects of Systems Control.

11. Rice, Fred J.
"The State-of-the-Art and Control of Packet Switching and Networks"
IEEE ICC 75, Vol. 1, 6-1 to 6-5. This paper reviews the results of the on-going investigation to determine appropriate techniques for controlling packet switching networks.

12. Roberts, Lawrence G.
"Development of Packet Switching Networks Worldwide"
Telecommunications Oct 76. This article describes the techniques that will be employed to establish a worldwide switching network.

B. Design Methodology (continued)

13. Rosner, Roy Daniel
"Packet Switching and Circuit Switching: A Comparison"
NTC Dec 75, Vol. 1, 42-1 to 42-7. In this paper a large common user network for data communications is investigated in an operational environment.

14. Rosner, R. D., R. H. Bittel, and D. E. Brown
"A High Throughput Packet Switched Network Technique without Message Reassembly"
IEEE ICC 75, Vol. 1, 6-11 to 6-16. A new packet switching network technique is described which introduces a substantially different technique for handling traffic.

15. Thurber et. al.
"A Systematic Approach to the Design of Digital Bussing Structures"
Fall Joint Computer Conference, 1972, pp. 719-740.

16. Ulfers, Horst E.
"PACKNET — A Packet Switching Data Network Simulator"
IEEE ICC 75, Vol. 1, 6-22 to 6-28. This paper describes a discrete simulation model of a packet switched data network.

C. High Order Language Design

1. Cardenes et. al.
"Programming Languages"
Library of Congress 71-169162 Chapter 10, Sections 1-6, pp. 316-354

2. Carter, Lynn
"Microprocessor Programming Trade-offs; Which Language is Best?"
RDN, October 5, 1976. Discussion of the advantages in combining the use of assembly languages and high level language.

3. Davis, Zucker, Campbell
"A Building Block Approach to Multiprocessing"
Spring Joint Computer Conference, 1972, pp. 25-30.

4. Evans, Arthur and Robert C. Morgan
"Development on a Communications Oriented Language"
Report Number BBN 3261, Parts I and II. This document is a preliminary design of a Communications Oriented Language (COL).

5. Foster, James R., Jr., 1st Lt. USAF
"Development of a High Order Language Architecture"
NAECON "71" Record, pp. 201-205.

6. "High-Level or Low? Language Choice Hinges on μP-System Production Volume"
Digital Design, August 1976, pp. 25-30. Discussion of break points between high- and low-level language usage based on the production volume.

7. Kline, Barbara, et. al.
"The In-Circuit Approach to the Development of Micro-Computer-Based Products"
Proceedings IEEE, Vol. 64, No. 6, June 1976, pp. 937-942.

C. High Order Language Design (continued)

8. Mallett, Patrick W. and T. G. Lewis
"Considerations for Implementing a High Level Microprogramming Language Translation System" Computer, August 1975, pp. 40-52. This paper presents some considerations that affect the realization of a high-level microprogramming language translation system.

9. Stover, D. R.
"A Proposed Long Range HOL Plan for Collins Radio"
Collins Radio Internal Working Paper WP-3954, March 1976.

D. Standards

1. Directive 5000.29, Department of Defense.
Subject: "Management of Computer Resources in Major Defense Systems".

2. Horelick, Larsen
"CAMAC: A Modular Standard"
IEEE Spectrum, April 1976, pp. 50-55.

3. Jones, Louise H.
"A Survey of Current Work in Microprogramming"
Computer, August 1975, pp. 30-38. This article provides a brief survey of current work; it has good bibliography.

4. Knoblock, Loughry, Vissers
"Insight Into Interfacing"
IEEE Spectrum, May 1975, pp. 50-57

5. MIL-STD-1553A, 30 April 1975
"Aircraft Internal Time Division Command/Response Multiplex Data Bus".

6. Norr, W. M. (Naval Air Development Center, Warminster, Pennsylvania)
"Microprocessor Application and Standardization in Naval Avionics Systems" (Naval Air Systems Command, AIR-360, under AIR-TASK WF21-241-601). Some of the steps already taken within DoD and private industry are described, and the philosophy needed for standardization for cost reduction is presented.

7. Performance Specifications for TRI-TAC Unit Level Message Switch
TT-B1-1202-0031, 15 April, 1976.